AFRL-IF-RS-TR-2003-220 Vol 1 (of 2)
Final Technical Report
September 2003

COMPILER OPTIMIZATIONS FOR POWER-AWARE COMPUTING

Georgia Tech Research Corporation

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J870

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-220 Vol 1 (of 2) has been reviewed and is approved for publication.

APPROVED: /s/
RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR:

/s/
JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	SEPTEMBER 2003	Final Apr 00 – Jan 03

4. TITLE AND SUBTITLE
COMPILER OPTIMIZATIONS FOR POWER-AWARE COMPUTING

5. FUNDING NUMBERS
C - F30602-00-2-0564
PE - 62301E
PR - HPSW
TA - 00
WU - 08

6. AUTHOR(S)
Vincent J. Mooney III

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Georgia Tech Research Corporation
School of Electrical & Computer Engineering
777 Atlantic Drive
Atlanta Georgia 30332-0250

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2003-220 Vol 1 (of 2)

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Raymond A. Liuzzi/IFTB/(315) 330-3577/ Raymond.Liuzzi@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

This final report summarizes work done on the DARPA funded project "Compiler Optimizations for Power Aware Computing." Volume I addresses methodologies invented that can be categorized as software based approaches, hardware based approaches and combined software/hardware based approaches. One of the software based approaches, data remapping, showed a 3.1X energy*delay reduction on a realistic example. One of the hardware based approaches, frequency/voltage scaling of second-level memory, showed a 1.3X energy*delay reduction on a realistic example. A combination of data remapping and frequency/voltage scaling of second level memory showed a 2.6X reduction in energy*delay but also showed the lowest power (energy/time) of any of the approaches considered. Volume II addresses realization of the world's first Wearable Motherboard or an intelligent garment for the 21st Century. The motherboard provides an extremely versatile framework for the incorporation of sensing, monitoring, and information processing devices.

14. SUBJECT TERMS
Power-Aware Computing, Architecture, Hardware/Software, Compiler

15. NUMBER OF PAGES
35

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1 Introduction

This final report summarizes the work done on the project entitled "Compiler Optimizations for Power Aware Computing" (COPAC). COPAC was supported by DARPA contract # AFRL F30602-00-2-0564.

The project started on April 28, 2000. Highlights of our findings are listed in the following sections.

Enclosed with this report is a separate final report for a subproject started in February 2001 on the wearable motherboard.

2 Target and Sample 2.6X Reduction in Energy*Delay

Before presenting a sample highlight results, we first need to briefly describe the base case.

2.1 Base Architecture

The typical embedded system consists of a processor, off-chip memory and Printed Circuit Board (PCB) buses connecting the chips as shown in Figure 1. We set up a simulation environment to measure each component shown in Figure 1 accurately. For our processor model we used MARS, a cycle accurate Verilog model of 5-stage RISC architecture obtained from the University of Michigan[24]. MARS can run ARM instructions. Compared to the high-level (e.g., instruction-level) power estimation methods, executing a Verilog model is more accurate but also more slow. We simulated the MARS model executing our applications using Synopsys VCS and measured power using the Synopsys Power Compiler. To obtain an accurate transistor-level model of MARS, we synthesized MARS using a TSMC 0.25μ library.

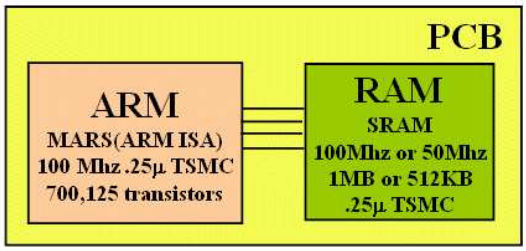


Figure 1: Architecture

To estimate memory power consumption, an analytical model based on D. Liu et al.[25] was used[4]. TSMC 0.25μ technology parameters and switching activity from VCS simulation were fed as input data. To estimate bus power, bus lengths on an actual board (Skiff board from HP) were measured and the capacitance and power were calculated [2, 6]. The compiler based technique is implemented on Trimaran, a compiler framework including ARM instruction generator. [4] explains the detail procedure of the power estimation.

In short, the base case, as shown in Figure 1, consists of (i) a 100MHz ARM-like RISC processor whose layout uses 700,125 transistors in TSMC 0.25μ technology, (ii) 1MB or 512KB of SRAM in a separate chip in TSMC 0.25μ technology running at either 100MHz or 50MHz, and (iii) a PCB with bus lines connecting the processor to the SRAM memory chip (second level memory).

2.2 Sample 2.6X Reduction in Energy*Delay Using Data Remapping and Frequency/ Voltage Scaling of Memory

The energy optimization methodologies we invented can be categorized as software based approaches, hardware based approaches and combined software/hardware based approaches. One software based approach is known as "Data Remapping" and is a software (compiler) technique[3, 5, 7]. This approach efficiently remaps an application's data layout in memory such that data elements that are accessed contemporaneously are also placed together in memory in contiguous address spaces. In such a way, data remapping reduces cache miss to the secondary memory since each load of a cache line (usually 4 or 8 or more words from contiguous memory addresses).

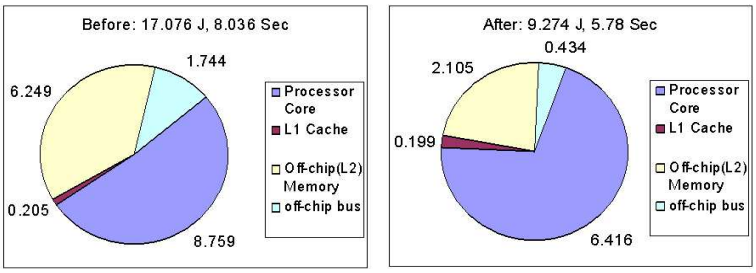


Figure 2: Energy distribution (DR+Freq/Volt Scale Mem., Health)

In this example we combined Data Remapping with a hardware-based approach, namely, voltage/frequency scaling of memory. We lowered supply voltage only for the off-chip memory while processor core kept the same voltage as the base case[2, 6]. We also adopted a store buffer to offset the performance degradation due to half-speed off-chip memory. The end result of the combination of these techniques can be seen in Figure 2. All four major consumers of energy shown on the left of Figure 2 for our base case of Figure 1 – the processor core, the processor's L1 cache, the off-chip (L2) memory and the off-chip buses – decrease when we apply Data Remapping and Frequency/Voltage Scaling of Off-Chip Memory, as shown on the right of Figure 2. Overall, energy decreases from 17.076 Joules to 9.274 Joules with a decrease in execution time as well: from 8.036 seconds to 5.78 seconds. The overall result achieved a 60.94% or 2.6X reduction in energy*delay [8].

3 Compiler/Architecture Savings

Figure 3 shows some representative highlights from the project. Figure 3 first shows some results from hardware based approaches, specifically, we highlight the idea we pioneered of lowering the voltage (and, unfortunately, such reduction necessitates the reduction of frequency as the achievable circuit speed reduces with reduced supply voltage) and frequency of off-chip L2 memory. The best result was a 28.51% (1.4X) reduction in energy based on this technique alone.

The middle section of Figure 3 highlights some results from software techniques. Unlike the previous hardware technique which decreases energy but unfortunately also increases execution time slightly, these software techniques decrease execution time and, as a result, also decreases energy (power consumption remains unchanged since no voltages vary at all)[3, 5, 7, 23]. While loop transformations did achieve up to a 50.63% (2X) reduction [11, 14], the best result here is from Data Remapping which achieved a 67.59% or 3.1X reduction in energy*delay. [3, 5, 7, 23].

The final highlight of Figure 3 was already discussed in Section 2.2.

One additional highlight is the discovery of the *Energy-Delay Ratio* paradigm[18, 19, 21, 22]. The EDR paradigm is based on a very interesting proof: energy is minimized in a VLSI system when energy consumption is proportional to delay for each logic unit. For example, if an adder has delay 10ns and a multiplier 60ns, then energy is minimized when the multiplier is designed such that it consumes 6X the amount of energy of the adder. Using the EDR paradigm, quick power-aware optimization decisions can be made at the transistor-, chip- and system-level.

Approach	Base Local	Goal System Level	Actual System Level	% Reduction Over Base	Base / Actual Sys. Level
HW Techniques					
Freq./Volt. Scale Mem(Fir)(mJ)	0.067	0.0067	0.0479	28.51	1.4X
Energy*Delay(mJ*ms)	0.002881	0.000288	0.002491	13.54	1.2X
Freq./Volt. Scale Mem(Perimeter)(J)	23.361	2.3361	17.86	23.55	1.3X
Energy*Delay(J*s)	248.9115	24.89115	191.8164	22.94	1.3X
SW Techniques					
DR+(Health)(J)	17.076	1.7076	9.274	39.33	1.6X
Energy*Delay(J*s)	137.2227	13.72227	44.4781	67.59	3.1X
Loop++(MXM)(J)	92.307	9.2307	64.914	29.68	1.4X
Energy*Delay(J*s)	14436.81	1443.681	7127.557	50.63	2.0X
Combined Technique					
DR+Freq./Volt. Scale Mem(Health)(J)	17.076	1.7076	9.274	45.69	1.8X
Energy*Delay(J*s)	137.2227	13.72227	53.60372	60.94	2.6X

Figure 3: Resulting Reductions in Energy and Energy*Delay

Many more results are available by perusing the over 20 publications resulting from this project. These publications are listed at the end of this report and are included in a CD provided with this report.

4 Additional Funding

4.1 HP Funding of Ozgur Celebican for Cycle-Accurate Energy Estimation of ARM Based Embedded Systems

Ozgur Celebican, a member of the Georgia Institute of Technology DARPA team under the supervision of PI Mooney, has been awarded by Hewlett-Packard funding for his graduate student stipend and tuition. Ozgur's project with HP is cycle-accurate energy estimation of ARM based embedded systems with I/O devices. Ozgur is in the process of modifying an ARM performance simulator to include I/O power simulation capabilities. Some of the research funded DARPA will is thus on a technology transition path for use in the design of commercial products for Hewlett-Packard.

4.2 NSF Funding of the EDR Paradigm

Co-PI Chatterjee has secured approximately $250,000 in additional funding from the National Science Foundation to continue his research on power optimization based on the EDR paradigm.

4.3 Intel Funding

Intel has granted some funds to Prof. Gao of the University of Delaware; Prof. Gao was a non-optional subcontract to this project.

5 Conclusion

In conclusion, new technology approaches have been discovered in frequency/voltage scaling of second-level memory and compiler optimizations including loop transformations and data-remapping. The best overall result as measured by energy*delay were a 3.1X reduction using data remapping. Also, a 2.6X reduction was achieved when

combining data-remapping and frequency/voltage scaling of memory; this example also achieved the lowest power (energy/time) consumption,

This first DARPA project for the PI was an exciting and exhilarating adventure in research collaboration, graduate student exhortation to achieve results and industry cooperation to find technology transfer paths both for those results as well as for the trained graduate students coming out of the project!

References

[1] L. Chakrapani, P. Korkmaz, V. Mooney, K. Palem, K. Puttaswamy and W. Wong, "The Emerging Power Crisis in Embedded Processors: What Can a (Poor) Compiler Do?" *Proceedings of Compilers, Architecture, and Synthesis for Embedded Systems (CASES'01)*, pp. 176-180, November 2001.

[2] K. Puttaswamy, L. Chakrapani, K. Choi, Y. Dhillon, U. Diril, P. Korkmaz, K. Lee, J. Park, A. Chatterjee, P. Ellervee, V. Mooney, K. Palem and W. Wong, "Power-Performance Trade-offs in Second Level Memory used by an ARM-Like RISC Architecture," in *Power Aware Computing*, Rami Melham and Bob Graybill, eds., pp. 211-226, New York: Kluwer Academic/Plenum Publishers, May 2002.

[3] K. Palem, R. Rabbah, V. Mooney, P. Korkmaz and K. Puttaswamy, "Design Space Optimization of Embedded Memory via Data Remapping," *Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems and Software and Compilers for Embedded Systems (LCTES/SCOPES)*, 2002, pp 28-37, June 2002.

[4] P. Korkmaz, K. Puttaswamy and V. Mooney, "Energy Modeling of a Processor Core Using Synopsys and of Memory Hierarchy Using Kamble and Ghose Model," CREST TR-02-002, February 2002.

[5] K. Palem, R. Rabbah, V. Mooney, P. Korkmaz and K. Puttaswamy, "Power Optimization of Embedded Systems via Data Remapping," GIT-CC-02-011, Georgia Institute of Technology, Feb. 2002

[6] K. Puttaswamy, K. Choi, J.C. Park, V. Mooney, A. Chatterjee and P. Ellervee, "System level power-performance trade-offs in embedded systems using voltage and frequency scaling of off-chip buses and memory," *Proceedings of the International Symposium on System Synthesis (ISSS'02)*, pp. 225-230, October 2002.

[7] R. Rabbah and K. Palem, "Data Remapping for Design Space Optimization of Enbedded Memory Systems," will appear in a special issue of the *ACM Transactions in Embedded Computing Systems*, vol. 2, No. 2, May 2003.

[8] S. Srinivasan, J.C. Park and V. Mooney, "Combining Data Remapping and Voltage/ Frequency Scaling of Second Level Memory for Energy Reduction in Embedded Systems," *Proceedings of the First International Workshop on Embedded Systems Codesign (ESCODES'02)*, pp. 57-62, September 2002.

[9] R. Banakar, M. Ekpanyopang, K. Puttaswamy, R. Rabbah, M. Balakrishnan, V. Mooney and K. Palem "An Access based Energy Model for the Datapath and Memory Hierarchy of HPL-PD Microarchitecture in Trimaran Framework (TRIREME)," Technical Report GIT-CC-02-35, Georgia Institute of Technology, June 2002.

[10] J.C. Park, V. Mooney, K. Palem and K. Choi, "Energy Minimization of a Pipelined Processor Using a Low Voltage Pipelined Cache," *Proceedings of 36th Annual Asilomar Conference on Signals, Systems and Computers*, pp. 67-73, November 2002.

[11] H. Yang, G. Gao, A. Marquez, G. Cai and Z. Hu, "Power and Energy Impact by Loop Transformations," *Workshop on Compilers and Operating Systems for Low Power*, in conjunction with *Parallel Architecture and Compilation Techniques (PACT'01)*, September 2001.

[12] R. Govindarajan, H. Yang, J. Amaral, C. Zhang and G.Gao, "Minimum Register Instruction Sequence Problem: Revisiting Optimal Code Generation for DAGs," *Proceedings of the International Parallel and Distributed Processing Symposium*, San Francisco, April, 2001.

[13] H. Yang, R. Govindarajan, G. Gao, G. Cai and Z. Hu, "Exploiting Schedule Slacks for Rate-Optimal Power-Minimum Software Pipelining," *Workshop on Compilers and Operating Systems for Low Power 2002*, in conjunction with *Parallel Architecture and Compilation Techniques (PACT'02)*, September 2002.

[14] H. Yang, G. Gao, C. Leung, R. Govindarajan and H. Wu, "On Achieving Balanced Power Consumption in Software Pipelined Loops," *Proceedings of the International Conference on Compilers, Artchitecture and Synthesis for Embedded Systems (CASES) 2002*, pp. 210-7, October 2002.

[15] D. Fan, H. Yang, G. Gao and R. Zhao, "An Evaluation of Different Branch Predictors for Low-power Embedded Processor," *Cool Chips V*, Tokyo, Japan, April 2002.

[16] H. Yang, R. Govindarajan, G. Gao and K. Theobald, "Power-Performance Trade-offs for Energy-Efficient Architectures: A Quantative Study," *Proceedings of the International Conference on Computer Design (ICCD) 2002*, pp. 174-179, September 2002.

[17] K. Choi and A. Chatterjee, "Efficient Instruction Level Optimization Methodology for Low-Power Embedded Systems," *Proceedings of the International Symposium on System Synthesis (ISSS'01)*, pp. 147-152, October 2001.

[18] K. Choi and A. Chatterjee, "Hierarchical power optimization for SoC (system-on-chip) through CMOS technology scaling," GIT-CC-02-03, Technical Report in Georgia Tech, 2002.

[19] A. Diril, Y. Dhillon, K. Choi and A. Chatterjee, "An O(N) Supply Voltage Assignment Algorithm for Low-Energy Serially Connected CMOS Modules and a Heuristic Extension to Acyclic Data Flow Graphs," *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03)*, pp. 173-179, February 2003.

[20] R. Govindarajan, H. Yang, J.Amaral, C. Zhang and G.Gao, "Minimum Register Instruction Sequencing to Reduce Register Spills in Out-Of-Order Issue Superscalar Architectures," *Proceedings of the IEEE Transactions on Computers*, Vol. 52, Issue 1, pp. 4-20, January 2003.

[21] K. Choi and A. Chatterjee, "HA2TSD: Hierarchical time slack distribution for ultra-low power CMOS VLSI," *Proceedinigs of the International Symposium on Low Power Electronics and Design*, pp. 207-212, September 2002.

[22] K. Choi and A. Chatterjee, "PA-ZSA (Power Aware Zero Slack Algorithm): A graph based timing analysis for ultra low-power CMOS VLSI," *Proceedings of the International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'02)*, pp. 178-187, October 2002.

[23] K. Palem and R. Rabbah, "Design Space Optimization of Embedded Memory Cache Systems via a Compiler," *Proceeding of the Sixth Annual High Performance Embedded Computing Workshop*, pp. 79-81, September 2000.

[24] The SimpleScalar-Arm Power Modeling Project, *http://www.eecs.umich.edu/~jringenb/power/*

[25] D. Liu and C. Svensson, "Power Consumption Estimation in CMOS VLSI Chips," *Proceedings of the IEEE Journal of Solid-State Circuits*, Vol. 29, No. 6, pp. 663-670, June 1994.

Appendix A:

System Level Power-Performance Trade-Offs in Embedded Systems Using Voltage and Frequency Scaling of Off-Chip Buses and Memory

Kiran Puttaswamy[1], Kyu-Won Choi[2], Jun Cheol Park[1],
Vincent J. Mooney III[1], Abhijit Chatterjee[2] and Peeter Ellervee[3]
[1]Center for Research in Embedded Systems and Technology
[1,2]School of Electrical and Computer Engineering
Georgia Institute of Technology
{kiranp, kwchoi, jcpark, mooney, chat}@ece.gatech.edu
[3]Tallinn Technical University
lrv@cc.ttu.ee

ABSTRACT

In embedded systems, off-chip buses and memory (i.e., L2 memory as opposed to the L1 memory which is usually on-chip cache) consume significant power, often more than the processor itself. In this paper, for the case of an embedded system with one processor chip and one memory chip, we propose frequency and voltage scaling of the off-chip buses and the memory chip and use a known micro-architectural enhancement called a store buffer to reduce the resulting impact on execution time. Our benchmarks show a system (processor + off-chip bus + off-chip memory) power savings of 28% to 36%, an energy savings of 13% to 35%, all while increasing the execution time in the range of 1% to 29%. Previous work in power-aware computing has focused on frequency and voltage scaling of the processors or selective power-down of sub-sets of off-chip memory chips. This paper quantitatively explores voltage/frequency scaling of off-chip buses and memory as a means of trading off performance for power/energy at the system level in embedded systems.

Keywords

Voltage/Frequency Scaling, Power-Performance Trade-offs, Embedded Systems, Design Space.

1. INTRODUCTION

A typical embedded system consists of at least three main components: a processor (often with L1 cache), an off-chip memory (called L2 memory) and an off-chip bus connecting the processor and memory. The off-chip components, being highly capacitive, may consume as much or more power than the processor. This suggests that we can gain significant reductions in power and energy by reducing the off-chip voltage and frequency. However, power reduction from voltage (and corresponding frequency) reduction could be compromised by an increase in execution time, thus resulting in overall increase in energy dissipation (note that the increase in execution time is due to increased memory access time and is a function of the cache misses). We demonstrate how the performance impact of voltage and frequency scaling can be reduced by implementing a known micro-architectural technique called a store buffer. While our approach can apply to dynamic voltage scaling, this paper only shows the tradeoffs between statically setting the off-chip bus and memory voltage at 3.3 Volts and frequency at 100 MHz versus 2 Volts and 50 MHz. As is evident from the above description, it is necessary to have an integrated framework in order to quantitatively explore the power-performance design space at the system level. Specifically, the contribution of this paper is as follows.

We have combined the techniques of frequency/voltage scaling of off-chip buses and memory (circuit level technique) with a store buffer (architectural technique) to realize reductions in both system power and system energy dissipation with a negligible impact on the execution time.

The rest of the paper is organized as follows: Section 2 discusses the motivation for this work. Section 3 gives an overview of the previous work. Section 4 describes the experimental infrastructure. Section 5 discusses the methodology. Section 6 presents the results and Section 7 concludes the paper.

2. MOTIVATION

In embedded systems, especially mobile applications, battery life is a significant concern. It has been established that the battery drains faster when power is drawn at a higher rate [1]. For example, a battery which lasts for 1000 hours when drawing 10 milliamps at 1.5 Volts will only last for 80 hours when drawing 100 milliamps at the same voltage [1]. Also, an old battery discharges faster than a new one. Currently, few hooks exist to trade-off performance for power to prolong the battery life of an embedded system. For example, when the user is executing a time-critical application like real-time video-conferencing, he might decide to operate at peak performance and high power. Or he might opt for low performance and very low power when executing low priority applications like checking e-mail. Or he might chose an intermediate power-performance point based on the existing battery capacity. Note that turning off memory chips may not be possible because, for example, video data might be stored in the memory and may

ISSS'02, October 2–4, 2002, Kyoto, Japan.

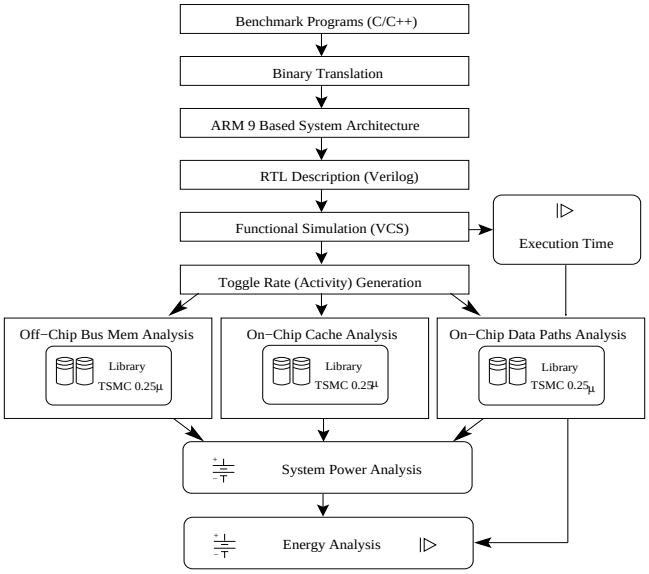


Figure 1: Experimental Infrastructure

need to be used. Also note that, as presented in this paper, this is a static technique and happens at the beginning of the program execution. We currently do not address dynamic (run-time) scaling of voltage/frequency of off-chip memory. This paper presents an approach that can allow the compiler and/or the user to decide at which power-performance point to operate, e.g., based on the knowledge of battery capacity and battery discharge pattern.

3. PREVIOUS WORK

There has been significant work in the field of voltage and frequency scaling. Voltage scaling techniques have been investigated at almost all levels of the design hierarchy from the system level to the device level due to the quadratic effect on the switching power dissipation. However, as the supply voltage becomes lower, the circuit delay increases and the performance degrades [2]. Techniques to improve performance fall into three main categories: reducing the threshold voltage to improve circuit speeds, introducing parallelism into the architecture while using slower device speeds, and using multiple supply voltages to choose the lowest supply voltage for different circuit components that still satisfies the speed requirements. Our approach falls into the third category.

At the system/architecture level, a number of memory optimization schemes for low power have been developed [3, 5]. Briefly, those approaches can be categorized as follows: cache optimization, memory access reduction (especially for off-chip memory), memory sizing/structuring and memory-intensive voltage scaling. Our work falls in the category of memory intensive voltage scaling.

There has also been a lot of work on system level power analysis for software power dissipation. In [6], profiling hardware is used to identify tightly coupled regions of code and dynamically optimize the configuration of the microprocessor so as to minimize performance penalty. Our work is similar to this approach in the sense that we set the performance parameters, namely, voltage and frequency based on the requirements of the application. However, note that ours is a static technique rather than a dynamic technique. Also, there has been work on dynamically adjusting the speeds (i.e., either lower the frequency or shutdown the unused

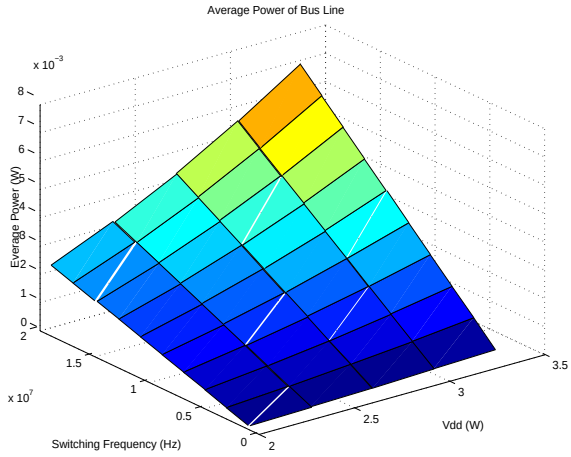


Figure 2: Bus Power Dissipation Pattern

modules), whichever gives the best results [7]. At the technology level, various techniques have been researched such as gating the supply voltage to cache memories [14].

A similar framework to our power measurement infrastructure is introduced in SimplePower [4]. SimplePower is based on a subset of the Simplescalar Instruction Set Architecture. Currently, Simplepower does not capture the energy consumed by the control unit of the processor nor the clock generation nor the distribution network [13]. On the other hand, our work more accurately models the specific RISC processor as the measurements are based on cycle-accurate functional simulations and Register Transfer Level hardware models used along with an actual technology library. An additional aspect of our work is the inclusion of power models for both off-chip memory [19] and the Printed Circuit Board bus.

4. EXPERIMENTAL INFRASTRUCTURE

We consider an embedded system which consists of a classic five stage pipeline RISC processor core with 4 kilobytes of instruction cache, 4 kilobytes of data cache, a single off-chip synchronous SRAM memory of size 0.5 Megabytes organized as 128K X 4 bytes, and a bus interface consisting of a 32 bit address bus, 32 bit data bus and the read/write control signals between the processor core and the SRAM memory.

Our experimental infrastructure (Figure 1) consists of four main components: a C Compiler; MARS – obtained from the University of Michigan – a cycle-accurate Verilog Model of a RISC processor capable of running ARM instructions [12]; a power model for off-chip buses; and a power model for memory.

We use the GNU-gcc ARM cross compiler version egcs-2.91.66. For each benchmark we consider, we compile the benchmark to relocatable ARM assembly code using GNU-gcc ARM cross compiler. Then we use the GNU cross-assembler to generate a binary executable targeted towards ARM architectures. Then we translate the binary into an ascii format called VHX (Verilog HeX) [20] which is suitable for being simulated on MARS using the Synopsys VCS simulator [8]. The simulation experiments are carried out in two modes. In the first mode, the CPU core and off-chip buses and memory are all operating at 100 MHz, a setup that is very similar to a hardware setup we have in the Hewlett-Packard "Skiff" Personal Server board [17] with a StrongARM SA-110 processor and 16 megabytes of off-chip memory [18]. (Note that we model a smaller off-chip memory of size 0.5 megabytes in accordance with the smaller applications we consider.) We obtained the simulation

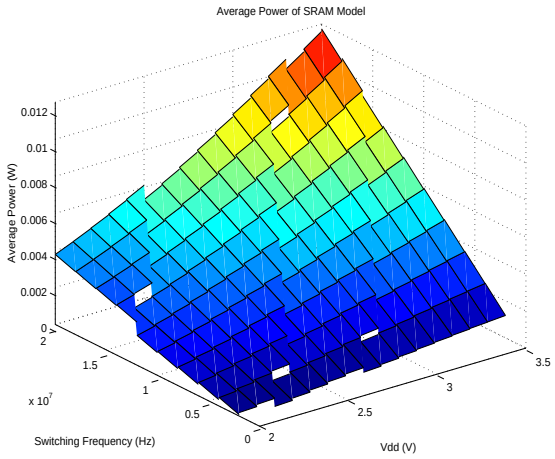


Figure 3: Memory Power Dissipation Pattern

model for the off-chip memory from IDT Technologies, Inc. [15]. In the second mode, the processor core includes a Verilog description of a store buffer integrated into the core and interfaced to off-chip buses and off-chip L2 memory operating at 50 MHz. In both cases, using the set of benchmarks in Table 1, each benchmark is simulated and switching activity is collected for the processor core, off-chip buses and off-chip memory models. The switching activity is fed to the power models of the core, off-chip buses and off-chip memory along with the technology parameters of a TSMC 0.25μ CMOS technology standard cell library from Leda Systems [9] to obtain power and energy estimates.

4.1 On-chip Datapath Power Estimation

We use a synthesis based methodology for developing the power models for the submodules belonging to the datapath (we consider the datapath to consist of the fetch unit, decode unit, register file, arithmetic logic unit, data cache access unit and writeback unit). The synthesis infrastructure consists of two software tools from Synopsys, Inc.: the Design Compiler and Power Compiler [8]. Design Compiler generates the gate level netlist from the hardware description of the submodules, and Power Compiler generates power estimates for each of the synthesized netlists. The Verilog RTL description is given as input to the Synopsys Design Compiler. The output netlist is generated using a TSMC 0.25μ CMOS technology standard cell library from LEDA Systems [9]. The technology details include features such as transistor width, transistor length, gate capacitance, drain capacitance, transistor rise time and transistor fall time. The TSMC 0.25μ standard cell library is characterized for leakage power, thus enabling us to include both dynamic and static power and energy in our analysis.

The synthesis process was guided by fixing the maximum delay and maximum area. The maximum delay was set to 10 ns and the maximum area was fixed to infinity so as to get the fastest implementation. In our case, the modules were synthesized to operate at greater than or equal to 100 MHz (i.e., at less than or equal to a 10 ns cycle time).

We use Power Compiler from Synopsys to estimate the power of on-chip components. The Power Compiler obtains the switching activity of the various functional modules based on the simulation of benchmarks on MARS. Then, Power Compiler annotates this switching activity onto the synthesis environment and obtains estimates of the dynamic and static power dissipation for the particular technology chosen (in our case, TSMC 0.25μ CMOS technology).

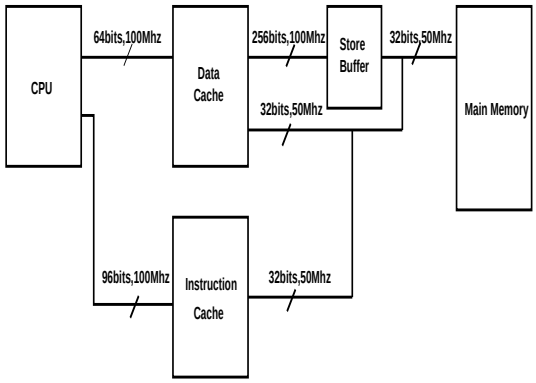


Figure 4: System Level Block Diagram

Benchmark	Size (bytes)	Explanation
bubble	8131	Bubble Sort Program
factorial	6641	Factorial Program
fi b	6815	Fibonacci Sequence Calculation
matmul	8058	Matrix Multiplication
sort_int	7241	Integer Array Sort

Table 1: Benchmarks

4.2 Off-chip Bus Power Analysis

We use Spectre simulation to obtain the power for off-chip buses. The driver component is modeled by a series of inverters (buffers) of increasing size and the model is designed using TSMC 0.25μ CMOS process technology parameters. The parameters of TSMC 0.25μ process technology are available through MOSIS [11]. The bus line capacitance values are obtained from actual measurement on a PCB board using the Intel StrongARM processor [18]. The details of the measurement procedure has been explained in [20].

The graph in Figure 2 describes the dependence of the power dissipation on the switching frequency of the bus and also the power supply voltage V_{dd}. The power dissipation is found to decrease as the switching frequency and the supply voltage are decreased. (Switching frequency is how often a signal actually switches values. Clearly, switching frequency of a signal is data- or profile-dependent, i.e., dependent on a particular benchmark and data.) As the supply voltage decreases, the average power dissipation reduces quadratically. As the switching frequency decreases, the average power dissipation decreases almost linearly. Note that the simultaneous halving of both voltage and frequency results in cubic savings.

4.3 Memory Power Analysis

We use an analytical SRAM model [19] for the off-chip memory and cache power dissipation. For the off-chip memory power model, we updated the analytical model [19] using the TSMC 0.25μ process technology parameters. We use the switching activity from simulations to obtain estimates for SRAM memory power dissipation. The variation of power for the memory with supply voltage V_{dd} and the switching frequency is thus found.

The graph in Figure 3 describes the dependence of the power dissipation of the memory with the power supply voltage V_{dd} and the switching frequency for a memory size of 0.5 megabytes (note that switching activity period is the reciprocal of switching frequency). As the supply voltage is decreased, the average power dissipation is found to decrease quadratically. Also the memory delay was found

to double as we reduced the voltage from 3.3 Volts to 2 Volts, reducing the maximum clocking frequency possible from 100 MHz at 3.3 Volts to 50 MHz at 2 Volts.

For the on-chip cache power dissipation model, we use the same SRAM model [19]. The model is combined with the capacitance values obtained from the TSMC 0.25μ CMOS technology parameters. Note that the main difference between the off-chip memory model and on-chip cache model is that the off-chip model has significantly higher capacitance values (due to size) than the on-chip model.

4.4 System Level Power/Energy Model

We define the sum total of the processor core power, bus power and the memory hierarchy power as the system power P_{sys}.

$$P_{sys} = P_{cpu} + P_{bus} + P_{mem} \quad (1)$$

P_{cpu} is the power dissipated by the processor core,
P_{bus} is the power dissipated by the off-chip buses,
P_{mem} is the power dissipated by the memory.

Also, we calculate the system energy E_{sys} for each benchmark by multiplying the execution time collected by simulation and the corresponding sysem power P_{sys}.

$$E_{sys} = P_{sys} * t \quad (2)$$

P_{sys} is the power dissipated by the system,
t is the total execution time of the benchmark.

5. METHODOLOGY

We now explain the method we use to explore voltage and frequency scaling as a static technique for design space exploration in terms of power versus performance trade-offs.

5.1 Voltage/Frequency Scaling

In particular, we analyze the case where we reduce the voltage of the off-chip memory and off-chip buses from 3.3 Volts to 2 Volts (note that our original system consists of the MARS [12] processor powered at 2.75 Volts with the memory buses and memory chip powered at 3.3 Volts). The resulting increase in memory delay (it doubles) is taken care of by reducing the off-chip bus and off-chip memory frequency from 100 MHz to 50 MHz. Of course, reducing off-chip bus and off-chip memory frequency increases program execution time in proportion to cache misses. We help offset this effect by adding a store buffer to the processor model. The store buffer helps to reduce cache miss penalties generated due to "store" instructions (since the cache model follows a read-write allocation policy, cache miss servicing takes up a significant amount of time). On a cache miss on store, the processor stores the data into the store buffer. The store buffer assumes the responsibility of flushing the data into the main memory.

We achieve frequency scaling by simulating the off-chip components at the lower frequency of 50 MHz. We achieve voltage scaling by using the resulting switching activity with the reduced voltage value of 2 Volts to calculate the off-chip bus and memory power dissipation values.

5.2 Store Buffer Technique

Figure 4 shows an architectural description of the embedded system with a store buffer included. Note that the store buffer is a 16 entry buffer. Stores to external memory are first placed in the store buffer and subsequently taken out when the off-chip bus is available. Thus, we reduce cache miss latency due to "store" instructions. The store buffer maintains synchronization with the on-chip data cache as well as with the off-chip memory. We use the Synopsys Power Compiler to estimate the additional power consumed due to the addition of store buffer into the processor core. Since we have integrated the description of the store buffer into MARS, the new system level power dissipation includes the store buffer overhead.

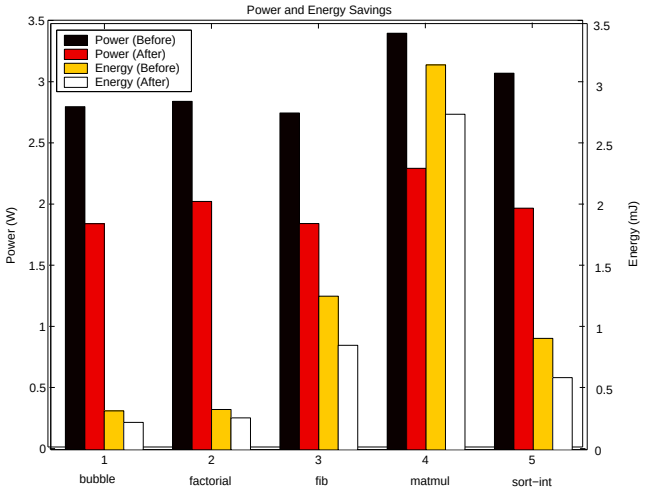


Figure 5: Power/Energy Saving Results

5.3 Design Space Exploration

In this section, we discuss the design space exploration of the power-performance space using voltage and frequency scaling. The Verilog model (MARS) is simulated in two modes. In the first mode, processor, off-chip buses and off-chip memory operate at 100 MHz. In the second mode, the processor operates at 100 MHz while the off-chip buses and off-chip memory operate at 50 MHz. In both cases, the Power Compiler collects the switching activity and uses the collected switching activity to give an estimate of dynamic and static power dissipation of all the modules within the core. We also collect the switching activity of off-chip memory elements and off-chip buses and feed the collected switching activity information to the analytical models to obtain the power estimates for off-chip bus and off-chip memory. We obtain the system level power/energy estimates as explained in Section 4.4.

Our calculations do not include the extra overhead of multiple supply voltage generation since we assume that the board already has the multiple supply voltages needed. For example, the "Skiff" Personnal Server Board from HP/Compaq [18] has 2 Volt, 3.3 Volt and 5 Volt power supplies. Also, currently there are memory chips from NEC Semiconductors where the chip component operates at 3.3 Volts and the I/O buffer component can operate at either 3.3 Volts or 2.5 Volts [16]. For example, the memory chip μPD4442361 Synchronous SRAM can operate at 3.3 Volt chip core voltage and either 3.3 Volt or 2.5 Volt I/O buffer voltage. The μPD4442361 is available in three speed grades of 133 MHz, 117 MHz and 100 MHz with the corresponding access times of 6.5 ns, 7.5 ns and 8.5 ns [16]. Therefore, we think it is reasonable to assume that our system will have at least three supply voltages readily available and that the system memory chips are capable of operating at dual voltages. The choice of frequency of off-chip buses and memory is currently assumed to be made statically by a mechanical or electrical switch.

benchmark	Executable size (kB)	Dynamic instruction count	Input data size	Data cache accesses	Data cache misses	Data cache miss %
bubble	34.852	7503	50 integers array	1675	107	6.39
factorial	34.634	6033	1 integer	2006	250	12.46
fi b	34.651	30602	1 integer	11840	262	2.21
matmul	34.857	21642	0.5 kB	7358	4916	66.81
sort_int	34.763	23171	0.5 kB	7808	104	1.33

Table 2: Execution Statistics for Various Benchmarks

	Off-chip Bus, Memory at 100 MHz, 3.3 V				Off-chip Bus, Memory at 50 MHz, 2 V				% Improvement
Benchmark	cpu+cache (W)	bus (mW)	L2 memory(mW)	Total (W)	cpu+cache(W)	bus(mW)	L2 memory(mW)	Total (W)	
bubble	1.24	301.64	1276.49	2.817	1.22	96.14	541.08	1.857	34.07
factorial	1.18	444.35	1236.96	2.861	1.15	93.16	797.08	2.040	28.69
fi b	1.25	287.68	1228.23	2.766	1.25	92.50	516.06	1.859	32.79
matmul	1.07	637.48	1713.34	3.421	1.04	129.04	1143.51	2.313	32.39
sort_int	1.27	336.78	1485.92	3.093	1.27	111.91	604.11	1.986	35.79

Table 3: System Level Power Estimates

6. RESULTS

The benchmarks in Table 1 are chosen to be computationally intensive. Also, the size of the data has been suitably modified so as to generate a significant number of L1 cache misses as can be seen from Table 2. For example, with matmul we increased the array sizes so that the 4KB L1 data cache could not hold the working set. Some of these benchmarks constitute the kernel of many signal-processing algorithms.

Table 2 shows the dynamic instruction count, cache access and miss statistics for the given benchmarks. Note that the final miss rates are smaller than the average miss rates at the beginning/middle of the execution of the program due to the temporal and spatial locality of the cache memories. Also note that the matmul benchmark has a very high miss rate. As a direct consequence of this, this benchmark experiences high off-chip traffic. As we will see later, benchmarks such as this, which need high off-chip bandwidth, show correspondingly lower improvement in terms of energy by our technique (although they still benefit in terms of power) as the power savings are nullified by the high increases in program execution time.

Table 3 presents the results related to system level power. The first three columns indicate the three components of power, namely, core power dissipation, off-chip bus power dissipation and off-chip memory power dissipation for the case where the MARS processor core operates at 2.75 Volts and 100 MHz and the off-chip system operates at 3.3 Volts and 100 MHz. The next three columns indicate the same three components of power for the case where the off-chip bus and memory operate at 2 Volts and 50 MHz while the MARS processor core still runs at 2.75 Volts and 100 MHz. The system level power estimate is obtained by adding up the core power, bus power and the memory power as shown in the fifth and ninth "Total(W)" columns in Table 3. Note that the proposed technique has reduced the power dissipation by an average of 32% on the five benchmarks. Also note that the average power reduction is almost uniform across all benchmarks irrespective of their execution characteristics like execution time and dynamic instruction mix. This highlights the dominant effect of voltage and frequency on the power dissipation.

Table 4 presents the statistics for the architectural and circuit level design space exploration where execution time represents the performance axis and system power represents the power axis. Note that, as expected, matmul benchmark has a higher penalty in terms of the execution time due to high off-chip traffic for loading and storing the data arrays.

The energy column combines both the design space axes, namely performance axis and the power axis and serves as a baseline for analyzing power-performance trade-offs. The trade-off shown, for example, the factorial benchmark, shows a performance penalty of 11.37% in return for a power reduction of 28.69% and an energy reduction of 20.48%. All benchmarks show improvements in both power and energy. However, as expected, the execution time increases. This shows that our technique reduces both power and energy by virtue of reducing the fraction of the off-chip bus and memory power consumed, at a (possibly small) penalty in increased execution time.

Figure 5 presents the overall results of the system level power and system level energy. Our technique of static voltage/frequency scaling combined with the store buffer is seen to reduce both power and energy at the system level.

7. CONCLUSION AND FUTURE WORK

In conclusion, we have shown how a simple trick of cutting off-chip voltage from 3.3 Volts to 2 Volts (note that this also cuts the voltage for the processor I/O pad driver logic), together with the enabled (due to extra latency available) reduction in frequency of off-chip buses and memory, reduces both power and energy. The basic point is that both the compiler and the programmer can take advantage of smart architectural and memory hierarchy features, which allow the reduction of power with some corresponding trade-offs in terms of performance. For example, the choice of 100 MHz versus 50 MHz for off-chip bus frequency could be made dynamically programmable (e.g., by writing to a special on-chip register or a memory-mapped location). In this case, then, code could be written which operates at high performance and high power during critical times, but scales down to lower performance and low power during non-critical time periods. This paper lays the groundwork for such a system where off-chip buses and memory and possibly more peripherals have their power dissipations modulated either by the user or by the compiler.

We will look at further hiding the load-instruction memory access latency caused by slowing down the off-chip buses and off-chip memory. We will explore other configurations for off-chip memory and also other memory technologies (e.g., SDRAMs). We will pursue making the voltage and frequency scaling dynamic.

	Off-chip Bus, Memory at 100 MHz, 3.3 V			Off-chip Bus, Memory at 50 MHz, 2 V			Percent Change	
Benchmark	Execn Time (μs)	Power (W)	Energy (mJ)	Execn Time (μs)	Power (W)	Energy (mJ)	Execn Time increase (%)	Energy decrease (%)
bubble	113.945	2.817	0.321	122.265	1.857	0.227	7.3	29.3
factorial	116.115	2.861	0.332	129.325	2.040	0.264	11.37	20.48
fib	456.795	2.766	1.263	463.245	1.859	0.861	1.4	31.83
matmul	924.735	3.421	3.164	1192.98	2.313	2.759	29.0	12.8
sort_int	296.425	3.093	0.917	300.265	1.986	0.596	1.29	35.0

Table 4: System Level Design Space Exploration

8. ACKNOWLEDGEMENTS

This research was funded by DARPA under contract number F30602-00-2-0564. We also acknowledge donations received from Cadence, Hewlett-Packard, Intel, LEDA Systems, Mentor Graphics, Sun and Synopsys.

9. REFERENCES

[1] P. Horowitz and W. Hill, *The Art of Electronics*, Second Edition, Cambridge University Press, England, 1989.

[2] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, Vol. 27, April 1992.

[3] T. Ishihara and K. Asada, "A System Level Memory Power Optimization Technique using Multiple Supply and Threshold Voltages," *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 456-461, June 2001.

[4] W. Ye, N. Vijaykrishnan, M. Kandemir and M. J. Irwin, "The Design and Use of Simplepower: A cycle-Accurate Energy Estimation Tool," *Proceedings of 38th Design Automation Conference*, pp. 340-345, June 2000.

[5] L. Benini, A. Macii, E. Macii, and M. Poncino, "Synthesis of Application-Specific Memories for Power Optimization in Embedded Systems," *Proceedings of 38th Design Automation Conference*, pp. 300-303, June 2000.

[6] A. Iyer and D. Marculescu, "Power Aware Microarchitecture Resource Scaling," *Proceedings of Design Automation and Test in Europe*, pp. 190-196, March 2001.

[7] A. Acquaviva, L. Benini and B. Ricco, "An Adaptive Algorithm for Low-Power Streaming Multimedia Processing," *Proceedings of Design Automation and Test in Europe*, pp. 273-279, March 2001.

[8] Synopsys, Inc., Available HTTP: *http://www.synopsys.com*

[9] LEDA Systems, Inc., Available HTTP: *http://www.ledasys.com*

[10] TSMC, "IP Services," Available HTTP: *http://www.tsmc.com/design/ip.html*

[11] The MOSIS Service, Available HTTP: *http://www.mosis.org*

[12] The SimpleScalar-Arm Power Modeling Project, *http://www.eecs.umich.edu/~jringenb/power/*

[13] The SimplePower Energy Estimation Tool, *http://www.cse.psu.edu/~mdl/SimplePower.html*

[14] M. Powell, S. Yang, B. Falsafi, K. Roy and T. N. Vijaykumar, "Gated V_{dd}: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 90-95, July 2000.

[15] IDT Technologies, Inc., Available HTTP: *http://www.idt.com/products/pages/ZBT_Verilog_p.html*

[16] NEC Semiconductors, Inc., Available HTTP: *http://www.ic.nec.co.jp/memory/english/products/sram/ssram-4m.html*

[17] Hewlett-Packard, Inc., Available HTTP: *http://www.hp.com*

[18] HP Labs - Cambridge Research Laboratory Personal Server Project, Available HTTP: *http://crl.research.compaq.com/projects/personalserver/personal-server-spec.html*

[19] D. Liu and C. Svensson, "Power Consumption Estimation in CMOS VLSI Chips," *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 6, pp. 663-670, June 1994.

[20] P. Korkmaz, K. Puttaswamy and V. Mooney, "Energy modeling of a Processor core using Synopsys and of the Memory Hierarchy using the Kamble and Ghose Model," Technical Report, CREST-TR-02-002, Georgia Institute of Technology, Feb. 2002.

Appendix B:

Combining Data Remapping and Voltage/Frequency Scaling of Second Level Memory for Energy Reduction in Embedded Systems

Sudarshan K. Srinivasan, Jun Cheol Park and Vincent J. Mooney III
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA-30332
{darshan, jcpark, mooney}@ece.gatech.edu

Abstract

In this paper we show that the energy reductions obtained from using two techniques, data remapping and voltage/frequency scaling of off-chip bus and memory, combine to provide interesting trade offs between energy, execution time and power. Both methods aim to reduce the energy consumed by the memory subsystem. Data remapping is a fully automatic compile time technique applicable to pointer-intensive dynamic applications. Voltage/frequency scaling of off-chip memory is a technique applied at the hardware level. When combined together, energy reductions can be as high as 46% .The improvements are verified in the context of two OLDEN pointer-centric benchmarks, namely Perimeter and Health.

1 Introduction

In embedded systems, memory is a significant power/energy sink, often consuming as much as half of the total power/energy [2]. In this paper we focus on simultaneously applying a hardware technique and a compile time technique in order to obtain significant energy savings. Our target processor is an ARM-like processor. The two methods we apply are data remapping (compile time technique) and voltage/frequency scaling of the off-chip bus and memory.

An embedded system usually consists at least of a processor (including L1 cache), off-chip memory and off-chip bus. The off-chip components are highly capacitive. This causes them to consume close to half of the digital system power (where digital system power is the power consumed by the processor plus memory). Slowing down the off-chip memory by scaling the voltage and frequency[19, 20, 15] can be used to reduce the energy consumed by the off-chip memory. But slowing down the off-chip memory will also reduce performance.

Data remapping[11, 10] is a compile time technique. It is used to remapp the application's data layout so that data elements that are accessed contemporaneously are placed together in memory. Remapping improves spatial locality and thus reduces cache misses. Cache misses are expensive in terms of performance. Remapping leads to a reduction in the execution time and energy.

The main drawback of the voltage/frequency scaling technique is the reduction in performance. Combining the two techniques can increase the overall reduction in energy. Furthermore, where execution time allocated is fixed, the combination of faster execution due to data remapping can offset the slower execution time due to reduced clock frequency for off-chip memory resulting in the same original execution time at dramatically reduced power (and energy). Section 2 described related work. Section 3 describes the experimental infrastructure used for estimating the power and energy of the system for the before and after cases. Section 4 gives an overview of the data remapping algorithm. Section 5 describes the methodology used for voltage and frequency scaling, and Section 6 gives our design space exploration. Section 7 describe the results obtained after applying the two methods in terms of energy savings, and Section 8 concludes the paper.

2 Related Work

A framework similar to our infrastructure is Simplescalar ARM. It is a framework for power and performance analysis. Another is SimplePower[14] which implements a subset of the instructions supported by Simplescalar.

Unlike Simplescalar, our model (MARS, introduced later) is at the RTL (Register Transfer Level) level and thus is more accurate.

With respect to hardware techniques there is a lot of work going on to gate supply voltage to cache memories[6, 12], dynamically adjust the frequency or shutdown unused modules[4].

Related work in data reorganization[7] propose automated field re-ordering that assigns temporally related fields to adjacent memory locations. But they offer only partial solutions as they do not consider fields between different instances of a record.

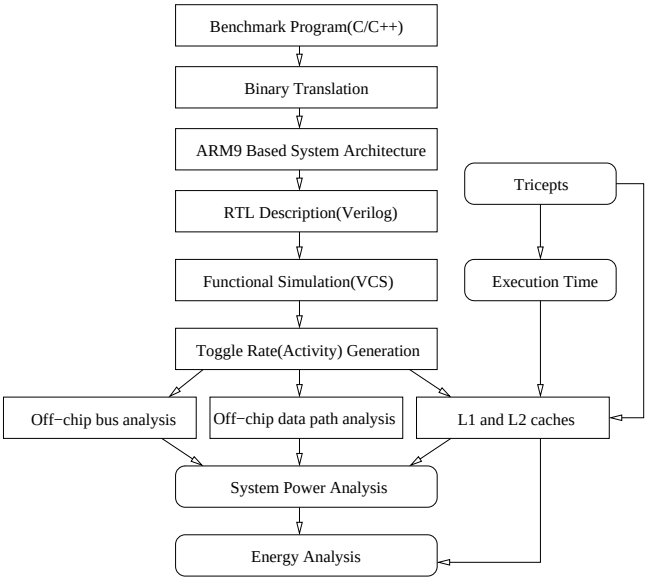


Figure 1: Experimental Infrastructure

3 Experimental Setup

In this section we describe the experimental setup used to simulate and evaluate the combined techniques of data remapping and voltage/frequency scaling.

The core of the simulation environment is MARS (Michigan Arm Simulator, obtained from University of Michigan)[16], capable of running ARM instructions. The power of the core processor can be estimated using Synopsys Power Compiler. The switching activity of the various nets is collected via simulation. The MARS model was synthesized using TSMC .25u library[18]. Using Synopsys Power Compiler[17], power models of the synthesized MARS model were created. The processor power was estimated using the power models and the switching activity of the nets at 2.75 volts.

The remapping benchmarks were implemented using TRIMARAN, a compiler framework (which includes the TRICEPS[8] ARM code generator and smart memory and cache hierarchy simulator (SMACHS)[21]. The execution statistics from TRIMARAN are used to estimate the power of the memory subsystem.

The model used has an L1 on-chip cache and L2 off-chip cache. To estimate the energy of the primary and secondary cache we assume an SRAM model. The Kamble and Ghose approach is used for energy estimation[9]. Execution statistics such as cache requests, read hits and misses, write hits and misses, execution cycles for both L1 and L2 cache are required. These execution statistics were obtained from simulations using SMACHS. Also, information about the cache configuration such as cache size, block size and tag size are required. One of the main disadvantages of the Kamble and Ghose method is that it does not model the I/O pads. Another disadvantage is that the model only accounts for dynamic power dissipation. This approximation (not including static/leakage power) is valid with respect to 0.25u technology but may not be valid for smaller(e.g., 0.09u) technologies.

The off-chip bus power is estimated using Spectre simulation. The driver component is modeled by a series of inverters. The model is designed using 0.25u TSMC library. The bus line capacitance values are obtained from actual measurements of a PCB with an Intel StrongARM1110 processor.

The total system power is estimated using the following approach. The energy for the L1 and L2 cache is obtained directly using the Kamble and Ghose model[15, 9]. The processor power is obtained from the Synopsys Power Compiler. Processor power is multiplied with the execution time to obtain the processor energy. The bus power obtained from the Spectre simulations is also multiplied with the execution time to obtain the energy of the off-chip bus. The energy from the bus, processor, L1 and L2 cache are summed up to get the total system power. The resulting measurements for our examples are shown in Section 7.

4 Data Remapping

Data Remapping is a compile time technique. It is an efficient remapping of an application's data layout in memory such that data elements that are accessed contemporaneously are placed together in memory. If a reference stream does not exhibit address adjacency, valuable resources are wasted as data is unnecessarily fetched and cached. The remapping technique remaps elements into new sets such that data items that are more likely to be used together are grouped together into the same cache block.

The applications to which data remapping can be usefully applied are record data type-heavy and pointer-heavy applications. Consider an example where in a file of records, a particular field of all records has to be searched or modified. The original mapping of the data in the memory will be such that fields belonging to a particular record will be placed together. If a cache line is fetched then all the data other than that particular field is wasted. Also the search for the next field will lead to a cache miss. Instead, if all fields were placed together in the above example then cache misses will be reduced. Also, energy is not wasted in fetching data that is not useful. The remapping algorithm is a combination of field reordering and customized placement to exhibit better spatial locality.

The remapping optimization consists of three phases – gathering phase, remapping of global data objects and remapping of dynamic data objects. In the gathering phase, an analysis of application memory access patterns along program hotspots[1] is performed. The remapping strategy cannot be arbitrarily applied to all data objects in the program. It is applied

based on the analysis obtained from the gathering phase. In the second phase global data objects are remapped. Once the candidate records have been identified, global program variables are filtered to isolate the arrays of records which are remapped. The third phase remaps dynamic data objects (i.e., pointer variables). The third phase is crucial as applications increasingly rely on dynamically allocated objects[3, 5].

5 Voltage and Frequency Scaling of Off-Chip Memory and Buses

The power consumed is proportional to the square of the voltage. Thus, reducing the voltage will lead to a quadratic reduction in power. But when the voltage of a component is lowered it leads to increase in delay which affects performance. The off-chip memory and buses are highly capacitive and thus consume close to half of the system power. To reduce the overall system power significantly we scale the voltage of the off-chip memory and buses. In our system the off-chip memory is an L2 cache.

Figure 2 shows the slowing down of L2 memory. The original system runs at 100MHz with the processor at 2.75 volts and off-chip components (including bus and memory) at 3.3 volts. The voltage of the off-chip bus and memory was scaled from 3.3 volts to 2 volts. This causes the delay of the off-chip memory and bus to double. To take into consideration the increase in delay, the frequency of the off-chip components was scaled from 100MHz to 50MHz.

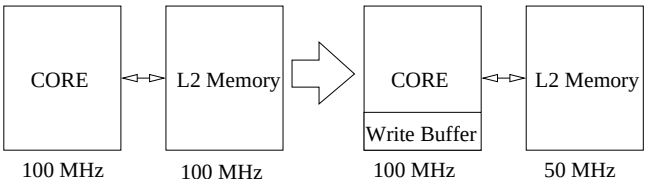


Figure 2: Slowing down L2 Memory

Frequency scaling is achieved by simulating the off-chip components at 50MHz instead of the original 100MHz. The D-Cache and I-Cache controllers were modified such that they fetch data from the memory at 50MHz instead of the previous 100MHz. This is done by doubling the latency of the memory (in our case the L2 cache) from 7 to 14 cycles. The voltage at which power is estimated is reduced from 3.3 volts to 2 volts to simulate voltage scaling in case of the off-chip bus and memory.

6 Design Space Exploration

The original system consists of the processor and off-chip components running at 100MHz. We simulate the system using two benchmarks health and perimeter before remapping. We call the original system the before case. The after case is where the processor is simulated at 100MHz and the off-chip components are simulated at 50MHz. The health and perimeter benchmarks are remapped and simulated with 50MHz L2 memory so that effect of combining both the techniques can be determined. Switching activity files are collected from the simulations using the MARS model and are used to determine the processor and bus power in both cases. The execution statistics from the Trimaran ARM simulator is used to determine the power for the L1 cache and off-chip L2 cache.

The data remapping allows the program to achieve the same overall execution time with half the cache resources. Since the cache is expensive in terms of both power and cost, halving the cache size would lead to roughly half the cost and power requirements. Results have been obtained by using half the L1 and L2 cache size.

The power calculations do not include the overhead of multiple supply voltages as we assume that multiple supply voltages are already present in the board. Also it is assumed that voltage scaling (i.e., changing the frequency of off-chip components from 3.3 volts to 2 volts) is done statically.

7 Results

The energy savings from combining the two techniques has been shown for two Olden benchmarks[13], namely perimeter and health. The benchmarks selected are such that they are suitable for remapping. The perimeter allocates quad trees at the program startup and do not modify them. The primary data structure used in health is a link list to which elements are added and removed.

Table 1 shows the results for the health benchmark. The L1 is a 32KB cache with 16 bytes line size. The L2 is a 1MB cache with 32 bytes line size. We find that for the health benchmark there is a large reduction in the execution cycles, but for perimeter the reduction in execution cycles is not as much (see Table 2. Data remapping will cause an increase in performance but much of this performance gain is lost due to slowing down the off-chip memory. This is clearly seen in the case of the Health benchmark. Also we find that for both benchmarks there is a large decrease in the energy of the L2 cache. Even though the processor power is almost constant, the decrease in processor energy is due to gains in performance due to remapping. We are able to achieve a maximum of 46% energy gains in the Health benchmark. From our experiments, we observed that there is no simple linear relationship among data remapping, frequency/voltage scaling of second level memory, energy reduction and energy delay reduction.

The remapping technique allows a program to run with the same execution time but with far less the memory. To explore the design space, we also considered reducing the L1 and L2

Table 1: Energy Delay with Frequency/Voltage Scaling of Memory (FVM) and Data Remapping(DR) for Health Benchmark

	Before DR, FVM	After DR	After FVM	After DR+FVM	After DR+FVM 1/2size L1	After DR+FVM 1/2size L2	After DR+FVM 1/2size L1,L2
Execution Cycles	803645821	479612138	892552982	578046486	603275469	711151104	736311686
Delay(Execution Time)(s)	8.036	4.796	8.926	5.780	6.033	7.112	7.363
Energy(J)	17.076	9.274	14.316	9.274	9.468	11.158	10.134
Energy*Delay	137.231	44.479	127.778	53.608	57.118	79.350	74.618
% Energy Reduction	0.00	39.33	16.16	45.69	44.55	34.66	40.65
% Energy*Delay Reduction	0.00	67.59	6.89	60.94	58.38	42.18	45.63

Table 2: Energy Delay with Frequency/Voltage Scaling of Memory (FVM) and Data Remapping(DR) for Perimeter Benchmark

	Before DR, FVM	After DR	After FVM	After DR+FVM	After DR+FVM 1/2size L1	After DR+FVM 1/2size L2	After DR+FVM 1/2size L1,L2
Execution Cycles	1065497740	967549770	1073983968	999065267	999305168	999095525	999339410
Delay(Execution Time)(s)	10.655	9.675	10.740	9.991	9.993	9.991	9.993
Energy(J)	23.361	21.648	17.860	16.897	16.414	14.221	13.828
Energy*Delay	248.911	209.455	191.814	168.812	164.026	142.081	138.189
% Energy Reduction	0.00	7.33	23.55	27.67	29.74	39.13	40.81
% Energy*Delay Reduction	0.00	15.85	22.94	32.18	34.10	42.92	44.48

Table 3: Energy results after remapping and Voltage Scaling(L1=32KB, L2=1MB) for Health Benchmark

Execution Cycles	Processor Energy(J)	Off-chip Bus Energy(J)	L1 Cache Energy(J)	L2 Cache Energy(J)	Total Energy(J)	% reductions Total Energy
578046486	6.415	0.433	0.3155	2.104	9.274	45.69

cache sizes to half their original sizes. The last three columns in Tables 1 and 2 show the energy results after halving L1 cache, L2 cache and both L1 and L2 cache respectively. We find that as expected the energy requirements of the cache also reduced by half. In case of the Perimeter benchmark the execution time remains the same and thus the energy saving in the memory subsystem is reflected in the overall energy gains. A maximum of 40.81% energy reduction is achieved in case of Perimeter benchmark when both caches are reduced to half their size. But in case of the Health benchmark, reduction in cache size leads to increase in the execution time. Even though the energy requirement of the memory subsystem is reduced, this is not reflected in the overall energy gains due to the increase in execution cycles. Thus, for the Health benchmark, the maximum energy reduction of 45.69% is found with both caches at their original sizes (L1=32KB, L2=1MB).

8 Conclusion

There are many techniques at both the hardware and compiler level aimed at saving energy and power. In this work we have demonstrated a combination of two techniques, one at the hardware level and one at the compiler level. The main drawback of hardware techniques is that they tradeoff power with performance. In our work by combining the two techniques, we are able to obtain energy gains without leading to a performance loss. For future work, we are looking at additional architecture level techniques aimed at the memory subsytem (specifically at the cache) and processor where compiler and hardware techniques interact to reduce energy.

9 Acknowledgements

This research was funded bt DARPA under contract number F30602-00-2-0564. We acnowledge help received from Rodric Rabbah in running the Trimaran simulations of the Health and Perimeter benchmarks. We also acknowledge donations received from Cadence, Hewlett-Packard, LEDA Systems, Mentor Graphics, Sun and Synopsys.

References

[1] T. Ball and J. Larus, ”Efficient path profiling,” *Proceedings of the 29th Annual International Symposium on Microarchitecture*, December 1996.

[2] P. Panda, N. Dutt and A. Nicolau, “Memory Issues In Embedded Systems-On-Chip, Optimizations and Exploration, ” Kluwer Academic Publishers, 1999.

[3] B. Calder, C. Krintz, S. John and T. Austin, ” Cache-conscious data placement,” *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems,* pp. 139-149, October 1998. Symposium on Theory of Computing, 1978.

[4] A. Acquaviva, L. Benini and B. Ricco, “An Adaptive Algorithm for Low-Power Streaming Multimedia Processing,” *Proceedings of Design Automation and Test in Europe*, pp. 273-279, March 2001.

[5] T. Chilimbi, M. Hill and J. Larus, ”Cache-conscious structure layout,” *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementatio,* pp. 1-12, May 1999.

[6] L. Benini, A. Macii, E. Macii and M. Poncino, "Synthesis of Application-Specific Memories for Power Optimization in Embedded Systems," *Proceedings of 38th Design Automation Conference*, pp. 300-303, June 2000.

[7] T. Chilimbi, B. Davidson and J. Larus, “Cache-conscious structure definition,” *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 13-24, May 1999.

[8] L. Chakrapani, K. Palem and W. Wong, “Enhancing the TRIMARAN compiler infrastructure for StrongARM code generation,” Technical Report CREST-TR-01-001, Georgia Institute of Technology, May 2001

[9] M. Kamble and K. Ghose ”Analytical energy dissipation models for low power caches,” *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 143-148, Aug. 1997.

[10] K. Palem, R. Rabbah, P. Korkmaz, V. Mooney and K. Puttaswamy, "Design Space Optimization of Embedded Memory Systems via Data Remapping," *Proceedings of the Languages, Compilers, and Tools for Embedded Systems (LCTES’02)*, pp. 28-37, June 2002.

[11] K. Palem and R. Rabbah. “Data remapping for design space optimization of embedded cache systems.” Technical Report GIT-CC-02-10, Georgia Institute of Technology, March 2002.

[12] T. Ishihara and K. Asada, "A System Level Memory Power Optimization Technique using Multiple Supply and Threshold Voltages," *Proceedings of 38th Design Automation Conference*, pp. 456-461, June 2001.

[13] OLDEN benchmark suite. *http://www.cs.princeton.edu/mcc/olden.html*.

[14] W. Ye, N. Vijaykrishnan, M. Kandemir and M. J. Irwin, "The Design and Use of Simplepower: A Cycle-Accurate Energy Estimation Tool," *Proceedings of 38th Design Automation Conference*, pp. 340-345, June 2000.

[15] P. Korkmaz, K. Puttaswamy and V. Mooney, “Energy modeling of a Processor core using Synopsys and of the Memory Hierarchy using the Kamble and Ghose Model,” Technical Report, CREST-TR-02-002, Georgia Institute of Technology, February 2002.

[16] The SimpleScalar-Arm Power Modeling Project, *http://www.eecs.umich.edu/~jringenb/power/*

[17] Synopsys, Inc., *http://www.synopsys.com*

[18] LEDA Systems, Inc., *http://www.ledasys.com*

[19] K. Puttaswamy, K. Choi, J. C. Park, V. J. Mooney III, A. Chatterjee and P. Ellervee, “System Level Power-Performance Trade-Offs in Embedded Systems Using Voltage and Frequency Scaling of Off-Chip Buses and Memory,” *Proceedings of International Symposium on System Synthesis*, to appear, October, 2002, Kyoto, Japan.

[20] K. Puttaswamy, L. N. Chakrapani, K. W. Choi, Y. S. Dhillon, U. Diril, P. Korkmaz, K. K. Lee, J. C. Park, A. Chatterjee, P. Ellervee, V. Mooney, K. Palem and W. F. Wong, "Power-Performance Trade-Offs in second level memory used by an ARM-Like RISC Architecture," in the book *Power Aware Computing*, Rami Melhem and Robert Graybill, Eds., Kluwer Academic/Plenum Publishers, 2002.

[21] Trimaran, *http://www.trimaran.org*

Appendix C:

Power and Energy Impact by Loop Transformations

Hongbo Yang†, Guang R. Gao†, Andres Marquez†, George Cai‡, Ziang Hu†

† Dept of Electrical and Computer Engineering
University of Delaware
Newark, DE 19716
{hyang,ggao,marquez,hu}@capsl.udel.edu

‡ Intel Corp
1501 S. Mopac Expressway, Suite 400
Austin, TX 78746
george.cai@intel.com

Power dissipation issues are becoming one of the major design issues in high performance processor architectures.

In this paper, we study the contribution of compiler optimizations to energy reduction. In particular, we are interested in the impact of loop optimizations in terms of performance and power tradeoffs. Both low-level loop optimizations at code generation (back-end) phase, such as *loop unrolling* and *software pipelining*, and high-level loop optimizations at program analysis and transformation phase (front-end), such as *loop permutation* and *tiling*, are studied.

In this study, we use the Delaware Power-Aware Compilation Testbed (Del-PACT) — an integrated framework consisting of a modern industry-strength compiler infrastructure and a state-of-the-art micro-architecture-level power analysis platform. Using Del-PACT, the performance/power tradeoffs of loop optimizations can be studied quantitatively. We have studied such impact on several benchmarks under Del-PACT.

The main observations are:

- Performance improvement (in terms of timing) is correlated positively with energy reduction.
- The impact on energy consumption of high-level and low-level loop optimizations is often closely coupled, and we should not evaluate individual effects in complete isolation. Instead, it is very useful to assess the combined contribution of both, high-level and low-level loop optimizations.
- In particular, results of our experiments are summarized as follow:
  - Loop unrolling reduces execution time through effective exploitation of ILP from different iterations and results in energy reduction.
  - Software pipelining may help in reducing total energy consumption – due to the reduction of the total execution time. However, in the two benchmarks we tested, the effects of high-level loop transformation cannot be ignored. In one benchmark, even with software pipelining disabled, applying proper high-level loop transformation can still improve the overall execution time and energy, comparing with the scenario where high-level loop transformation is disabled though software pipelining is applied.
  - Some high-level loop transformation such as loop permutation, loop tiling and loop fusion contribute significantly to energy

reduction. This behavior can be attributed to reducing both the total execution time and the total main memory activities (due to improved cache locality).

An analysis and discussion of our results is presented in section 4.

1 Introduction

Low power design and optimization [8] are becoming increasingly important in the design and application of modern microprocessors. Excessive power consumption has serious adverse effects – for example, the usefulness of a device or equipment is reduced due to the short battery life time.

In this paper, we focus on compiler optimization as a key area in low-power design [7, 13]. Many traditional compiler optimization techniques are aimed at improving program performance such as reducing the total program execution time. Such performance-oriented optimization may also help to save total energy consumption since a program terminates faster. But, things may not be that simple. For instance, some of such optimization my try to improve performance by exploiting instruction-level parallelism, thus increasing power consumption per unit time. Other optimization may reduce total execution time without increasing power consumption. The tradeoffs of these optimizations remain an interesting research area to be explored.

In this study, we are interested in the impact of loop optimizations in terms of performance and power tradeoffs. Both low-level loop optimizations at code generation (back-end) phase, such as *loop unrolling* and *software pipelining*, and high-level loop optimizations at program analysis and transformation phase (front-end), such as *loop permutation* and *tiling*, are studied.

Since both high-level and low-level optimization are involved in the study, it is critical that we should use a experimental framework where such tradeoff studies can be conducted effectively. We use the Delaware Power-Aware Compilation Testbed (Del-PACT) — an integrated framework consisting of a modern industry-strength compiler infrastructure and a state-of-the-art micro-architecture level power analysis platform. Using Del-PACT, the performance/power tradeoffs of loop optimizations can be studied quantitatively. We have studied the such impact on several benchmarks under Del-PACT.

This paper describes the motivation of loop optimization on program performance/power in Section 2 and describing the Del-PACT platform in Section 3. The results of applying loop optimization on saving energy are given in Section 4. The conclusions are drawn in Section 5.

2 Motivation for Loop Optimization to Save Energy

In this section we use some examples to illustrate the loop optimizations which are useful for energy saving. Both low-level loop optimizations at the code generation (back-end) phase, such as *loop unrolling* and *software pipelining*, and high-level loop optimizations at the program analysis and transformation phase (front-end), such as *loop permutation*, *loop fusion* and *loop tiling*, are discussed.

2.1 Loop unrolling

Loop unrolling [17]intends to increase instruction level parallelism of loop bodies by unrolling the loop body multiple times in order to schedule several loop iterations together. The transformation also reduces the number of times loop control statements are executed.

2.2 Software pipelining

Software pipelining restructures the loop kernel to increase the amount of parallelism in the loop, with the intent of minimizing the time to completion. In the past, resource-constrained software pipelining [10, 16] has been studied extensively by several researchers and a number of modulo scheduling algorithms have been proposed in the literature. A comprehensive survey of this work is provided by Rau and Fisher in [15]. The performance of software pipelined loop is measured by II(*initiation interval*). Every II cycles a new iteration is initiated, thus throughput of the loop is often measured by the value of II derived from the schedule. By reducing program execution time, software pipelining helps reduce the total energy consumption. But, as we will show later in the paper, the net effect of energy consumption due to software pipelining also depends on high-level loop transformations performed earlier in the compilation process.

2.3 Loop permutation

Loop permutation (also called loop interchange for two dimensional loops) is a useful high-level loop transformation for performance optimization [19]. See the following C program fragment:

```
for (i = 0; i < M; i++) {
 for (j = 0; j < N; j++) {
  a[j][i] = 1;
 }
}
```

Since the array a is placed by row-major mode, the above program fragment doesn't have good cache locality because two successive references on array a have a large span in memory space. By switching the inner and outer loop, the original loop is transformed into:

```
for (j = 0; j < N; j++) {
 for (i = 0; i < M; i++) {
  a[j][i] = 1;
 }
}
```

Note that the two successive references on array a access contiguous memory address thus the program exhibits good data cache locality. It usually improves both the program execution and power consumption of data cache.

2.4 Loop tiling

Loop tiling is a powerful high-level loop optimization technique useful for memory hierarchy optimization [14]. See the matrix multiplication program fragment:

```
for (i = 0; i < N; i++) {
 for (j = 0; j < N; j++) {
  for (k = 0; k < N; k++) {
   c[i][j] = c[i][j] +
    a[i][k] * b[k][j];
  }
 }
}
```

Two successive references to the same element of a are separated by N multiply-and-sum operations. Two successive references to the same element of b are separated by N^2 multiply-and-sum operations. Two successive references to the same element of c are separated by 1 multiply–and-sum operation. For the case when N is large, references to a and b exhibits little locality and the frequent data fetching from memory results in high power consumption.

Tiling the loop will transforme it to:

```
for (i = 0; i < N; i+=T) {
 for (j = 0; j < N; j+=T) {
  for (k = 0; k < N; k+=T) {
   for (ii = i; ii < min(i+T, N); ii++) {
   for (jj = j; jj < min(j+T, N); jj++) {
   for (kk = k; kk < min(k+T, N); kk++) {
```

```
      c[ii][jj] = c[ii][jj] +
        a[ii][kk] * b[kk][jj];
    }
    }
    }
   }
  }
}
```

Notice that in the inner three loop nests, we only compute a partial sum of the resulting matrix. When computing this partial sum, two successive references to the same element of `a` are separated by T multiply-and-sum operations. Two successive references to the same element of `b` are separated by T^2 multiply-and-sum operations. Two successive references to the same element of `c` are separated by one multiply-and-sum operation. A cache miss occurs when the program execution re-enter the inner three loop nests after i, j or k is incremented. However, cache locality in the inner three loops is improved.

Loop tiling may have dual effects in improving total energy consumption: it reduces both the total execution time and the cache miss ratios – both help energy reduction.

2.5 Loop fusion

See the following program fragment:

```
for (i = 0; i < N; i++) {
  a[i] = 1;
}

for (i = 0; i < N; i++) {
  a[i] = a[i] + 1;
}
```

Two successive references to the same element of `a` span the whole array `a` in the code above. By fusing the two loops together, we can get the following code fragment:

```
for (i = 0; i < N; i++) {
  a[i] = 1;
  a[i] = a[i] + 1;
}
```

The transformed code has much better cache locality. Just like loop tiling, this transformation will reduce both power and energy consumption.

3 Power and Performance Evaluation Platform

It is clear that, for the purpose of this study, we must use a compiler/simulator platform which (1) is capable of performing loop optimizations at both the high-level and the low-level, and a smooth integration of both; (2) is capable of performing micro-architecture level power simulations with a quantitative power model.

To this end, we chose to use the Del-PACT(Delaware Power-Aware Compilation Testbed) – a fully integrated framework composed of SGI MIPSpro compiler retargeted to the SimpleScalar [1] instruction set architecture, and a micro-architecture level power simulator based on an extension of the SimpleScalar architecture simulator instrumented with the Cai/Lim power model [5, 4], as shown in Figure 1. The SGI MIPSpro compiler is an industry-strength highly optimizing compiler. It implements a broad range of optimizations, including inter-procedural analysis and optimization (IPA), loop nest optimization and parallelization (LNO) [18], and SSA-based global optimization (WOPT) [2, 11] at high level. It also has an efficient backend including software pipelining, integrated global and local scheduler(IGLS) [12], and efficient global and register allocators (GRA and LRA) [3]. The legality of loop nest optimizations listed in Section 2 depends on dependence analysis [20]. The SGI MIPSpro compiler performs alias and dependence analysis and a rich set of loop optimizations including

those we will study in the paper. We have ported the MIPSpro compiler to the SimpleScalar instruction set architecture.

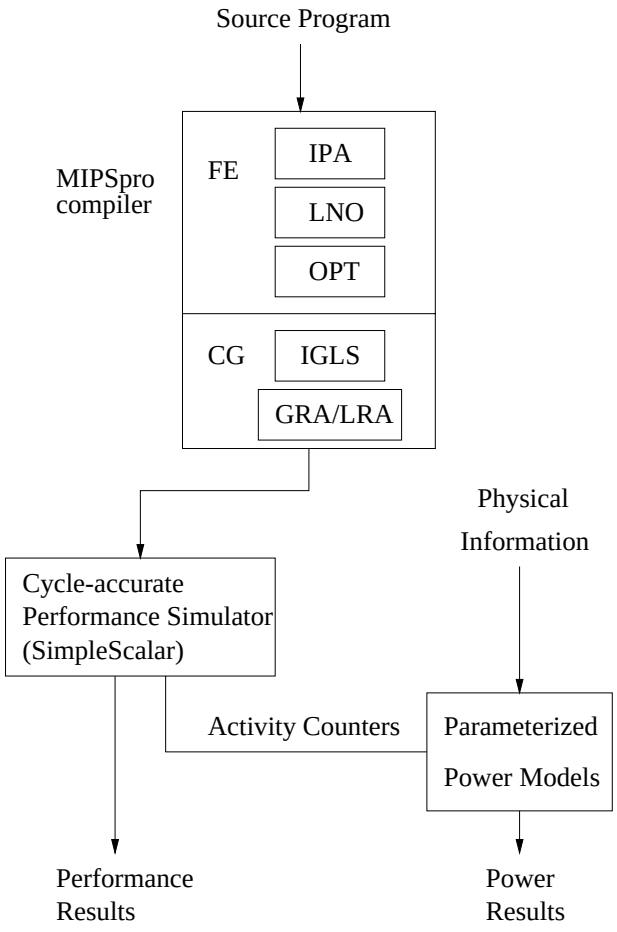


Figure 1: Power and Performance Evaluation Platform

The simulation engine of the Del-PACT platform is driven by the Cai/Lim power model as shown in the same diagram. The instrumented SimpleScalar simulator generates performance results and *activity counters* for each functional block. The physical information comes from approximation of circuit level power behaviors. During each cycle, the parameterized power model computes the present power consumption of each functional unit using the following formula:

$$\text{power} = AF * PDA * A + \text{idle power} + \text{leakage power}$$

AF Activity factor

PDA Active power density

A Area

The power consumption of all functional blocks is summed up, thus obtaining the total power consumption.

Other power/performance evaluation platforms exist as well. A model worth mentioning is SimplePower [9]. In [9] loop transformation techniques are evaluated. In their framework, high-level transformation and low-level loop transformations are performed in two isolated compilers while in our platform these two are tightly coupled into a single compiler. The difference between these two power models are left to be studied and a related work is found in [6].

4 Experimental Results

In this section, we present the experiments we have conducted using Del-PACT platform. Two benchmark programs: *mxm* and *vpenta* from the SPEC92 floating point benchmark suite are used. We evaluated the impact on performance/power of loop nest optimizations, software pipelining and loop unrolling. Loop nest optimization is a set of high-level optimizations that includes loop fusion, loop fission, loop peeling, loop tiling and loop permutation. The MIPSpro compiler analyzes the compiled program by determining the memory access sequence of loops, choosing those loop nest optimizations which are legal and profitable. Looking through the transformed code, we see that the loop nest optimizations applied on *mxm* is loop permutation and loop tiling, while those applied on *vpenta* are loop permutation and loop fusion. Performance, power and energy results of these transformations on each benchmark are shown in Figure 2.

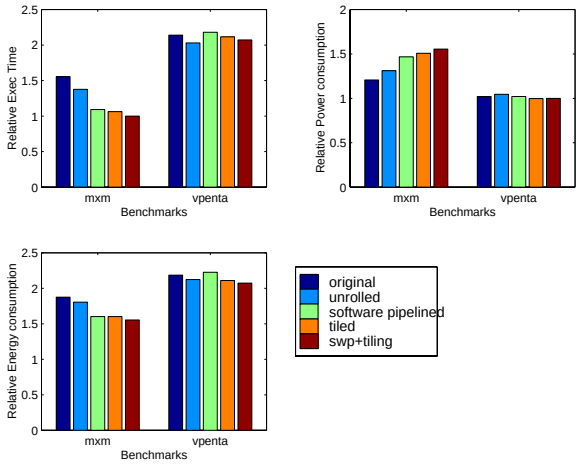


Figure 2: Performance, power and energy comparison

We observe that the performance improvement in terms of timing is correlated positively with the energy reduction. From Figure 2 we see the variation of execution time causes the similar variation in energy consumption. The results show that in the two benchmarks we have run, the dominating factor of energy consumption is the execution time.

Loop unrolling improves the program execution by increasing instructions level parallelism thus increasing power consumption correspondingly. For the *mxm* example, the instructions per cycle(IPC) increased from 1.68 to 1.8 by unrolling 4 times. For the *vpenta* example, loop unrolling reduces the total instruction count by 2% because of cross-iteration common subexpressions elimination. The IPC value before the loop unrolling and after that are 1.01 and 1.04 respectively.

Software pipelining helps in reducing total energy consumption in the *mxm* example. The IPC value without and with software pipelining are 1.68 and 1.9 respectively. Power consumption increase a little bit more as opposed to the case with loop unrolling because software pipelining exploits more instruction level parallelism than loop unrolling does. However, energy consumption is still reduced compared with the original untransformed program. In *vpenta* example, software pipelining does not help because of the high miss rate(13%) of level-1 data cache accesses.

Loop tiling and loop permutation applied on *mm* enhanced cache locality and they can improve the program performance more than software pipelining does. Loop permutation and loop fusion help the *vpenta* program reduce its level-1 data cache miss rate from 13% to 10%, thus reducing total energy consumption. Also these transformations make the performance improvement of software pipelining more evident compared with the case that software pipelining is applied without these high-level optimizations.

5 Conclusions

In this paper, we introduced our Del-PACT platform, which is an integrated framework that includes the MIPSpro compiler, SimpleScalar simulator and CAI/LIM power estimator. This platform can serve as the tool to make architecture design tradeoffs, and to study the impact of compiler optimization on program performance and power consumption. We use this platform to conduct experiments on the impact of loop optimizations on program performance vs power.

References

[1] Todd Austin. The simplescalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, Univ of Wisconsin, 1997.

[2] Fred Chow, Sun Chan, Robert Kennedy, Shin-Ming Liu, Raymond Lo, and Peng Tu. A new algorithm for partial redundancy elimination

based on SSA form. In *Proc. of the ACM SIGPLAN '97 Conf. on Programming Language Design and Implementation*, pages 273–286, Las Vegas, Nev., Jun. 15–18, 1997. *SIGPLAN Notices*, 32(6), Jun. 1997.

[3] Fred C. Chow and John L. Hennessy. The priority-based coloring approach to register allocation. *ACM Trans. on Programming Languages and Systems*, 12(4):501–536, Oct. 1990.

[4] A. Dhodapkar, C.H.Lim, and G.Cai. Tempest: A thermal enabled multi-model power/performance estimator. In Workshop on Power-Aware Computer Systems, Nov 2000.

[5] G.Cai and C.H.Lim. Architectural level power/performance optimization and dynamic power estimation. Cool Chips Tutorial, in conjunction with 32nd Annual International Symposium on Microarchitecture. Haifa, Israel, Nov 1999.

[6] Soraya Ghiasi and Dirk Grunwald. A comparison of two architectural power models. In Workshop on Power-Aware Computer Systems. Cambridge, MA, Nov 2000.

[7] Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Low power design methodologies: Hardware and software issues. Tutorial on Parallel Architecture and Compilation Techniques 2000.

[8] J.M.Rabaey and M. Pedram, editors. *Low-Power Design Methodologies*. Kluwer, 1996.

[9] Mahmut T. Kandemir, N. Vijaykrishnan, Mary Jane Irwin, and W. Ye. Influence of compiler optimizations on system power. In *Proceedings of the 37th Conference on Design Automation (DAC-00)*, pages 304–307, NY, June 5–9 2000. ACM/IEEE.

[10] Monica Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *Proc. of the SIGPLAN '88 Conf. on Programming Language Design and Implementation*, pages 318–328, Atlanta, Geor., Jun. 22–24, 1988. *SIGPLAN Notices*, 23(7), Jul. 1988.

[11] Raymond Lo, Fred Chow, Robert Kennedy, Shin-Ming Liu, and Peng Tu. Register promotion by sparse partial redundancy elimination of loads and stores. In *Proc. of the ACM SIGPLAN '98 Conf. on Programming Language Design and Implementation*, pages 26–37, Montréal, Qué., Jun. 17–19, 1998. *SIGPLAN Notices*, 33(6), Jun. 1998.

[12] Srinivas Mantripragada, Suneel Jain, and Jim Dehnert. A new framework for integrated global local scheduling. In *Proc. of the 1998 Intl. Conf. on Parallel Architectures and Compilation Techniques*, pages 167–174, Paris, Oct. 12–18, 1998. IEEE Comp. Soc. Press.

[13] M.Kandemir, N. Vijaykrishnan, M. J. Irwin, W. Ye, and I. Demirkiran. Register relabeling: A post-compilation technique for energy reduction. In *Workshop on Compilers and Operating Systems for Low Power 2000 (COLP'00)*.

[14] Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers Inc., 1997.

[15] B. R. Rau and J. A. Fisher. Instruction-level parallel processing: History, overview and perspective. *J. of Supercomputing*, 7:9–50, May 1993.

[16] B. R. Rau and C. D. Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific

computing. In *Proc. of the 14th Ann. Microprogramming Work.*, pages 183–198, Chatham, Mass., Oct. 12–15, 1981. ACM SIGMICRO and IEEE-CS TC-MICRO.

[17] Vivek Sarkar. Optimized unrolling of nested loops. In *Proceedings of the 2000 international conference on Supercomputing*, pages 153 – 166, Santa Fe, NM USA, May 8 - 11 2000.

[18] Michael E. Wolf, Dror E. Maydan, and Ding-Kai Chen. Combining loop transformations considering caches and scheduling. In *Proc. of the 29th Ann. Intl. Symp. on Microarchitecture*, pages 274–286, Paris, Dec. 2–4, 1996. IEEE-CS TC-MICRO and ACM SIGMICRO.

[19] Michael Wolfe. Advanced loop interchanging. In *Proc. of the 1986 Intl. Conf. on Parallel Processing*, pages 536–543, St. Charles, Ill., Aug. 19–22, 1986.

[20] Hans Zima and Barbara Chapman. *Supercompilers for Parallel and Vector Computers*. ACM Press, New York, 1990.

Appendix D:

HA²TSD: Hierarchical Time Slack Distribution for Ultra-Low Power CMOS VLSI

Kyu-won Choi and Abhijit Chatterjee
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332
{kwchoi,chat}@ece.gatech.edu

ABSTRACT
This paper describes an efficient hierarchical design and optimization approach for ultra-low power CMOS logic circuits. We introduce the *Hierarchical Activity-Aware Time Slack Distribution* (**HA²TSD**) algorithm, which distributes the surplus time slack into the most power-hungry modules hierarchically. **HA²TSD** ensures that the total slack budget is maximal and the total power is near-minimal. Based on these time slacks, we have optimized technology parameters (supply voltage, threshold voltage, and device width) through a gate-level power optimizer and have tested the algorithm on a set of benchmark example circuits and building blocks of a synthesizable ARM core. The experimental results show that our strategy delivers over an order of magnitude savings in total (static and dynamic) power and reduces the optimization run-time significantly.

Categories and Subject Descriptors
B.7.2 [Integrated Circuits]: Design Aids-simulation.
General Terms
Algorithms.
Keywords
Low-power design, time slack distribution, and gate-level power optimization.

1. INTRODUCTION
Recent advances in wireless networking technology and the rapid development of semiconductor technology have introduced new challenges in the design of portable devices such as personal digital assistants (PDAs). Power optimization for those embedded systems and power constrained mobile computing is an active area of research that has received considerable attention in most recent years. Delay, area and power trade-offs for complex systems require the use of advanced algorithms and EDA tools. To achieve excellent power and performance results, future EDA tools must harness the combination of technology parameters, i.e., multiple supply voltages (Vdd), multiple threshold voltages (Vth), and transistor resizing (W). By combining the optimization strategy with the on-the-fly technology parameter scaling, designers and EDA tools can fully explore the design space of dynamic power, static power, and timing slack [1,2].

In general, low-power optimizations that do not compromise performance are dependent on time slack calculation and the surplus delay (slack budget) distribution among the circuit modules. Time slack is measured as the difference between the signal required time and the signal arrival time at the primary output of each module. The first use of the slack distribution approach was reported by the popular *zero-slack algorithm* (**ZSA**) [3]. The **ZSA** is a greedy algorithm that assigns slack budgets to nets on long circuit paths. It ensures that the net slack budget is maximal, which means that no more slack budget can be assigned to any of the nets without violating the path timing constraints. Most other slack distribution algorithms are pruning versions of **ZSA** [4,5] for improving *delay performance of circuits*. However, the objective of the timing analysis in this paper is to provide a low-power methodology that maintains the high speed of circuits. The **HA²TSD** algorithm is different from the **ZSA** in three principal aspects: i) time slack distribution of each module is based on power rather than performance metrics; ii) the slack distribution is performed hierarchically, and iii) the technology parameters of each module are optimized at the gate level.

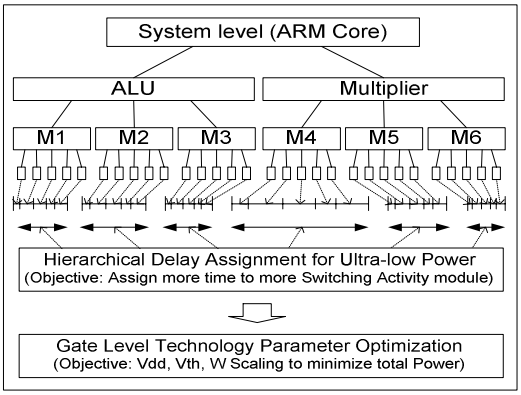


Figure 1. Hierarchical Delay Assignment and Gate Level Power Optimization

2. DELAY AND ENERGY MODEL
We use a transregional model for estimating the worst-case signal propagation delay through a gate. The delay model has been derived using an extension of the alpha-power law saturation drain current model [7] to the subthreshold region. The drain current model incorporates effects of high-field and quasi-ballistic (velocity overshoot) carrier transport in the MOSFET channel. The delay model consists of four major components: 1) the delay due to switching MOSFETs, 2) the distributed interconnect RC delay, 3) the time of flight delay, 4) the delay component due to the non-zero rise time of the input signal are considered. These definitions of gate delay and interconnect resistance delay allow the definition of arrival

ISLPED'02, August 12-14, 2002, Monterey, California, USA.

times and required times at the input and output of a gate in the network, which are used for defining time slack.

$$t_{d_v} = \left[\frac{1}{2} - \frac{1-\frac{V_{TS_v}}{V_{dd}}}{1+\alpha}\right] \max_{i\in(1,f_{ii}(v))}\{t_{d_{i,v}}\} + \frac{V_{dd}/2}{I_{Dvw} - f_{ii}(v)\beta I_{off}} \cdot \left[C_{DP_v} + \frac{1}{w_v}\sum_{j=1}^{f_{ii}(v)}(w_{vj}C_{t_{vj}} + C_{INT_{vj}})\right]$$

$$+ \max_{j\in(1,f_{oi}(v))}\{t_{d_{v,j}}\}\left[R_{INT_{vj}}(w_{vj}C_{t_{vj}} + \frac{1}{2}C_{INT_{vj}}) + \frac{L_{INT_{vj}}}{v_{INT}}\right] + \frac{1}{2}C_{m_v}V_{dd}\sum_{j=1}^{f_{ii}(v)-1}\frac{1}{I_{Dvw}(j)} \quad (1)$$

In the above equation, V_{dd} is the power supply voltage, t_{dv} is the delay of gate G_v, V_{TSi} is the threshold voltage of the ith gate, α is the velocity saturation coefficient ($1 \le \alpha \le 2$), $t_{di,v}$ is the delay of the gate G_v at the ith fan-in, $t_{dv,j}$ is the delay of the gate G_v at the jth fan-out, $I_{Dvw}(f_{ii})$ is the switching drain current per unit width, f_{ii} is the number of fanins, f_{oi} is the number of fanouts, β is the pMOSto nMOS width ratio ($\beta \ge 1$), I_{off} is the off current per unit width, C_{DPv} is the sum of the overlap, junction and finging capacitance at the output node per unit width, w_v is the device width, adjusting w_v scales the widths of all the transistors in G_v ($w_v \ge 1$), w_{vj} is the device width the gate at the jth fan-out ($w_{ij} \ge 1$), C_{tvj} is the input capacitance per unit width of the gate being driven by the jth fan-out, C_{INTvj} is the interconnect capacitance at the jth fan-out, R_{INTvj} is the interconnection resistance at the jth fan-out, L_{INTvj} is the interconnection length at the jth fan-out, v_{INT} is the propagation velocity through the interconnect, C_{mv} is the intermediate node capacitance of series connected MODFET's in multiple fan-in gates, f_c is the clock frequency, η_v activity factor of the gate output, and K_{SC} is the coefficient for short-circuit dissipation [8]. The models are described in detail in our previous work [6].

The equations used to compute the dynamic and static energy dissipations of a gate are described next. Similar models have been presented and analyzed in a recent work by [8]. It is assumed that the gates are simple multi-input gates with symmetric series or parallel pull-up and pull-down MOSFET configurations. Contributions of subthreshold leakage through the MOSFET channel as well as the leakage across the device drain junctions to static dissipation are included.

1) Static Dissipation of Gate G_v ($v \in N$):

$$E_{Static_v} = V_{dd}W_v I_{off} / f_c \quad (5)$$

2) Dynamic and Short-Circuit Dissipation of G_v

$$E_{Dynamic_v} = \frac{1}{2}\eta_v V_{dd}^2(1+K_{SC}) \cdot \left[w_v\{C_{DP_v} + (f_{ii}(v)-1)C_{m_v}\}\sum_{j=1}^{f_{ii}(v)}(w_{vj}C_{t_{vj}} + C_{INT_{vj}})\right] \quad (6)$$

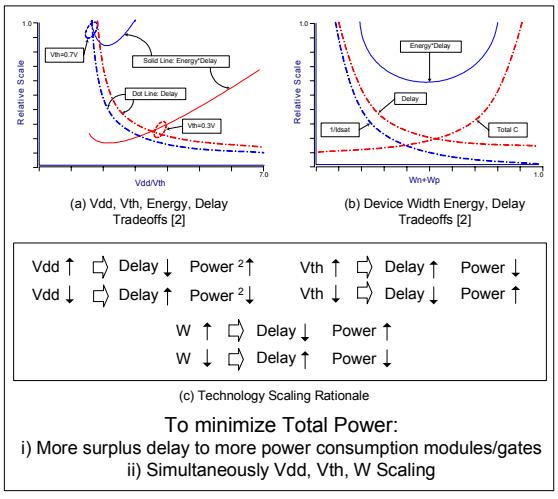
3. PREVIOUS WORK

Supply voltage scaling technique for low power has been investigated in almost all levels of the design hierarchy from system level to device level due to the quadratic effect on the switching power component. Many respective researches have been shown up in literature [1]. However, it does not come without penalties [9]. The scaling limitations of Vdd reduction are: 1) Delay increase (performance requirements impose a limit); and 2) Noise margins decrease (circuit is more susceptible to noise related soft failures). The approaches to overcome the extent of Vdd scaling are: 1) Availability of high-efficiency DC-DC converter for use [10]; 2) Scaling down the dimensions of devices along with Vdd to compensate for the effects of Vdd on performance; and 3) Reduction of the threshold voltage of transistors.

Threshold voltage scaling can be used to compensate the performance penalty of the Vdd reduction. In addition, for the active mode of operation, the low Vth is preferred because of the higher performance. However, for the standby mode, high Vth is useful for reduction of leakage power. Different threshold voltages can be developed by multiple Vth implantation during the fabrication, by changing the substrate and source bias, by controlling the back gate of double-gate SOI (silicon on insulator) devices [10]. Some techniques in literature are: 1) SATS (self adjusting threshold voltage scheme) [11]; 2) MTCMOS (multi-threshold voltage CMOS) [12]; 3) DTMOS (dynamic threshold voltage MOSFET) [13]; and 4) DGDT-SOI (double fate dynamic threshold control SOI) [14]. In general, the threshold voltage is a function of a number of parameters including the following: 1) Gate conductor, 2) Gate insulation material, 3) Gate insulator thickness-channel doping, 4) Impurities at the silicon-insulator interface, and 5) Voltage between the source and the substrate.

Transistor and gate sizing affects for dynamic and leakage power reduction and delay. A large gate is required to drive a large load capacitance with acceptable delay but requires more power. The basic rule is to use the smallest transistors or gates that satisfy the delay constraints. To reduce dynamic power, the gates that toggle with higher frequency should be made smaller. An interesting problem occurs when the sizing goal is to leakage power of a circuit. The leakage current of a transistor increases with decreasing threshold voltage and channel length. In general, a lower threshold or shorter channel transistor can provide more saturation current and thus offers a faster transistor. This presents a tradeoff between leakage power and delay. There have been a number of optimization algorithms for gate sizing for dozens of years [15].

Figure 2 presents the fundamental characteristics of those three device parameters (Vdd, Vth,W) for power and delay tradeoffs [2]. Figure 2(a) shows the Vdd/Vth and Delay*Energy tradeoffs. It shows that the supply voltage should be larger than four times of the threshold voltage if the delay is not to increase excessively. Figure 2(b) shows the Device Width and Delay*Energy tradeoffs. It is shown that the delay decreases with increase device width but the delay-energy product is minimized when the devices contribute half of the total load capacitance. The technology parameters trade-offs are summarized in Figure 2(c). In this paper, we try to optimize the non-linear parameters of those tradeoffs efficiently to minimize the total power.



(a) Vdd, Vth, Energy, Delay Tradeoffs [2]

(b) Device Width Energy, Delay Tradeoffs [2]

(c) Technology Scaling Rationale

Figure 2. Technology Optimization Rationale

4. PROPOSED APPROACH

The key steps of our approach are shown in Figure 3. First hierarchical circuit partitioning is performed. Then, beginning with the topmost level of the design hierarchy, delay values are assigned to every module at that level. The total delay from PI to PO is given. The problem is to determine the delays of the individual modules so that total power consumption can be minimized by optimizing the supply voltage, threshold voltage and device sizes of module M_j *for the assigned delay values*. The procedure is repeated hierarchically. We use the following heuristic to assign delays to each module.

Heuristic: In a given data flow graph of M_j modules, let $C_j = \sum_{node\ i} \eta_i c_i$ be the summation of the product of the activity η_i at node i and the capacitance c_i at node i over all nodes i of the module M_j. If the delay assigned to module M_j is D_j, then the best delay assignment for minimizing power is obtained when

$$\frac{D_1}{C_1} = \frac{D_2}{C_2} = \cdots = \frac{D_j}{C_j}$$

It is clear that such an assignment of delay to each M_j can cause the overall path delay constraint (sum of delays assigned to each module) to be violated for some of the paths in the module. Therefore, the iterative **HA^{2}TSD** algorithm is used to solve the problem. This is described below.

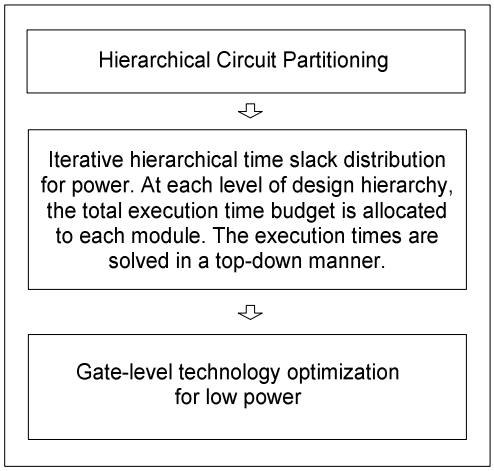


Figure 3. Power Optimization Procedure

4.1 Topological Depth-Based Partitioning

For simulation run-time efficiency and power optimization effectiveness, we introduce a circuit partitioning algorithm which ensures the minimization of the delay skew between sub-modules, and constrains maximum sub-module size (or fan-out size). Figure 4 gives conceptual overview of the topological depth-based partitioning. First of all, labeling of each circuit node is conducted according to the topological order. Then, according to the maximum depth and maximum size constraints, the whole flattened gate-level digital circuit is partitioned into sub-module circuits. The detailed algorithm for the partitioning is shown in Figure 5. The complexity of this algorithm is $O(b^m)$, where b is the branching factor (i.e., average fan-out number) and m is maximum topological depth.

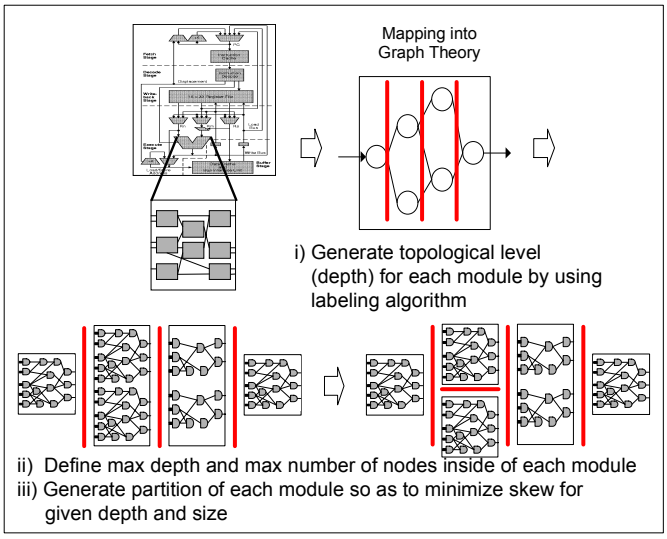


Figure 4. Partitioning Overview

HA2TSD Algorithm 1 : Partitioning

Input: *Directed acyclic graph* **G** = (**V,E**)
Output: *Partitioned sub-graphs* $G_i (V_i, E_i)$, *i= 2 … n*

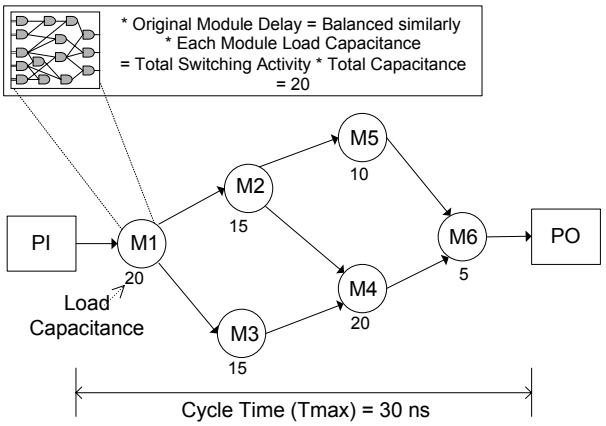
```
Begin
 Phase 0: Initialization
   Initialize the class variable of all nodes V in G ;
   (level (topological order) = ∞ )
 Phase I: Labeling {Identify topological order}
   Put a start node to FIFO Queue q;
   Initialize the level of the start node = 0;
   While (q Is not 0)
   {
       Obtain a reference to the first element(x) in q;
       Remove node x from q;
       For each fan-out node y of node x
       {
          If level of y = ∞ then
          {
             If number of fan-in node of y =1 then
                level of y = level of x +1;
             else if number of fan-in node of y >1 then
                level of y =MAX( levels of fan-in nodes of y) +1;
             Add node y into q;
          }
       }
   }
 Phase II: Partitioning
   Sort all nodes according to the topological order;
   Partition the graph G by constraints (max node number & depth);
End
```

Figure 5. Partitioning Algorithm

4.2 Activity-Aware Delay Assignment

Figure 6 presents an example of the module level delay assignment algorithm. In the first step, each module is sorted by the amount of load capacitance of each module (step 1). According to the priority of each module, we assign maximum delay with the "objective function" and "delay assignment" formula in Fig. 6 (Step 2 and 3). Then we look at the local improvement by local search (step 4). If all modules' delays are assigned, conduct the technology parameter optimization at the gate level (step 5). Finally, we generate the power/area saving values and optimal parameters. In the algorithm,

each module (***M1,…,Mi***) can be a functional module or a sub-partition, the total physical capacitance of a module can be the sum of the fan-in/out counts inside the module, and the load capacitance of each module can be calculated by multiplying the total switching activities by the total fan-in/out net counts. Its algorithm is shown in Figure 7. The complexity of the algorithm is $O(nb^m)$, where n is the number of modules, b is the branching factor (i.e., average fan-out number) and m is maximum topological depth.



Object : Assign max delay of each module for max power saving
(Note: slack time = power/area saving)

Object Function = $\frac{D_1}{C_1} = \frac{D_2}{C_2} = \cdots = \frac{D_6}{C_6}$

$$D_1 + D_2 + \cdots + D_6 \le T_{\max}$$

$$C_j = \sum_{node\ i} \eta_i c_i$$

(η_i = switching activity at node i, c_i = capacitance at node i)

Delay ssignment = $\frac{C_j}{\text{Total Load Capacitance Sum in Path}} \times (T_{\max}$ - Assigned Delay Sum)

Step 1: Module Priority queue for each module by load capacitance

M4, M1 (20)
M2 (15)
M3 (10)
M5,M6 (5)

Step 2: - Select M4
- Path Priority queue for each path with M4

Path1: M1,M2,M4,M6 (60)
Path2: M1,M3,M4,M6 (55)

- Select Path1
- Delay of M4 = (20/60)*30 = 10 ns

Step 3: - Repeat Step2 for all modules
- Delay of M1 = (20/40)*20 = 10 ns
- Delay of M2 = (15/20)*10 = 7.5 ns
- Delay of M3 = (10/15)*10 = 6.66 ns
- Delay of M5 = 10 ns
- Delay of M6 = 2.5 ns

Step 4: - Local search improvement
- Increase 6.66 to 7.5 for M3

Step 5: - Go to Gate level optimization (Vdd, Vth, W Scaling) with this Max delay of each module

Figure 6. An Example of Delay Assignment

HA2TSD Algorithm 2 : Delay Assignment

Input: *Partitioned sub-graphs* $G_i(V_i, E_i)$
Output: *Delay weighted sub-graphs* $G_i(V_i, E_i, W(v_i))$

Begin
Phase 0: *Initialization*
Enumerate the critical paths P_j *in* $G = \{G_1 \ldots G_i\}$;
Sort P_j *in decreasing order of criticality;*
Phase I: *Delay assignment for each path*
Identify maximum delay T_{max} *of all paths;*
Calculate switching activity α_i *for all nodes* V_i
Set the delay for nodes V_i *on critical path(s)*
While *(all path* P_j *)*
{
While *(unassigned* $V_i = 0$ *)*
{
$T_{max}(V_i) = [\alpha_i / (\alpha_1 + ..+\alpha_{i-1} + \alpha_{i+1}+..+ \alpha_n)] * T_{max}$;
/* where n = number of nodes on the P_j */
$W(V_i) = T_{max}(V_i)$;
}
}
End

Figure 7. Delay Assignment Algorithm

4.3 Gate-level Power Optimization

There are three ways to save power dissipation while maintaining operation frequency by utilizing surplus time slack in non-critical paths: i) employing multiple-Vdd to lower supply voltage, ii) employing multiple-Vth to reduce leakage current, and iii) employing multiple-W to reduce circuit capacitance. In this paper, *the Vdd reduction is main scaling parameter for low power, and Vth and W scaling is mainly for creating more time slack for the ultra-low power optimization*. The difficulties of the power optimization at gate level come from two major aspects: i) the non-linear interactions of the object parameters, for example, each gate has at least four non-linear variables (Vdd, Vth, W, Delay) and ii) the optimization time complexity, for example, after logic synthesis of target system, each functional module (i.e., ALU, Adder, Multiplier, etc.) might generate large number of gates/interconnections and the searching space for the optimization is exponential. Therefore, simulation-efficient partitioning scheme should be judiciously considered before the gate level optimization. The Figure 8 shows the relationship between the maximum delay assignment and the technology scaling for power savings.

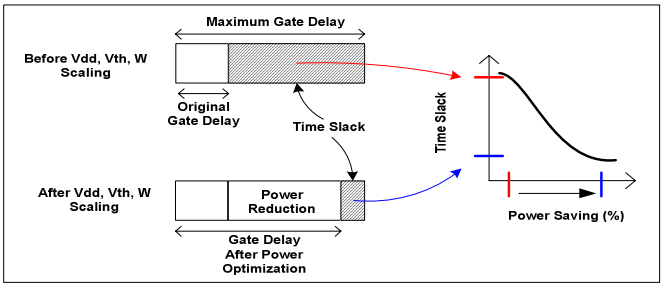


Figure 8. Time Slack and Power Saving

After the maximum delays have been assigned to each module/gate in the circuit, we optimize each gate individually for minimum power. The strategy is to find iteratively, using binary search, the optimal combination of Vdd, Vth, and W for each gate that meets the

maximum delay condition while achieving minimum power dissipation. We used our previous work for the gate level power optimization [6]. This strategy is based on the observation that power consumption and delay are monotonic functions of Vdd, Vth, and W, individually, other parameters being fixed. Since it is impractical to have more than one power supply or threshold voltage in the circuit, we keep only one global value of Vdd and Vth. However, the algorithm could be easily modified to allow the use of multiple threshold values in the circuit if desired. The algorithmic complexity of this procedure depends on the number of iteration steps that we allow for convergence to the optimal values. Assuming that V_{DD}, V_{th} and W are each constrained to 2^M quantized values, it takes $O(M^3)$ simulations of the entire circuit to obtain the final optimal values. This is many orders of magnitude lower than the complexity of any direct or random search algorithm that may be used to search for the optimal solution.

5. RESULTS

We developed a simulation frame work with C/C++/STL and Perl on Ultra-80 Unix machine for the hierarchical power optimization. Also, we used off-the-shelf commercial tools for the RTL description, the functional verification, and the logic synthesis of the target system. A few arithmetic modules from the target system and ISCAS89/MCNC91 benchmark circuits are used for the experimental demonstration. For the range of the technology parameter values, the 2001 updated version of ITRS (International Technology Roadmap for Semiconductors) and the MOSIS (Integrated Circuit Fabrication service) parameter test results with TSMC 0.25 micron are used. For the RTL design, we used verilog hardware description, for the functional simulation, we used VCS (synopsys), and for the logic synthesis, we used design analyzer (synopsys) with 0.25 micron TSMC library.

Monte Carlo simulation is performed for activity profiling of each module/sub-module as described in [2]. This approach consists of applying randomly generated input patterns at the primary inputs of the circuit and monitoring the switching activity per time interval T using a simulator. Under the assumption that the switching activity of a circuit module over any period T has a normal distribution, and for a desired percentage error in the activity estimate and a given confidence level, the number of required simulation vectors is estimated. The simulation based approach is accurate and capable of handling various device models, different circuit design styles, single and multi-phase clocking methodologies, tristate drives, etc.

Figure 9 shows the hierarchy and the granularity that we used in our simulation. In this paper, we only simulated 3-level hierarchical case. Table 1(a) shows the total power consumption with fixed technology parameters for the given circuits. Table 1(b) demonstrates the efficiency and effectiveness of the hierarchical power optimization with the proposed design flow. The experimental results show that our power optimization strategy delivers an order of magnitude savings in total (static and dynamic) power without performance degradation over non-optimized benchmark circuits and our hierarchical approach is much faster than traditional approach. With the hierarchical depth of 3 as shown in Figure 9, we can obtain average 6 times faster optimization than the totally flattened case when we still have average 83.6% power savings.

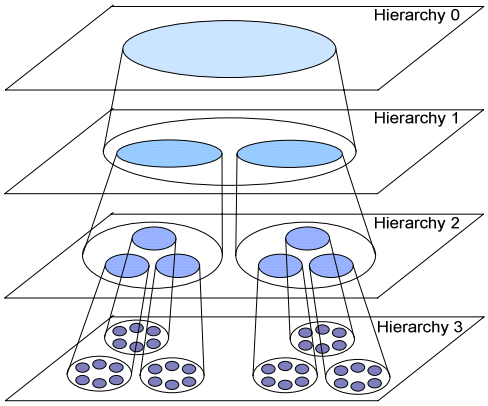


Figure 9. Hierarchy in our Simulation

6. CONCLUSION

This paper presents an efficient hierarchical low-power design flow and a novel switching activity based optimization algorithm for ultra-low power CMOS VLSI. Experimental results show that the algorithm yields reductions in power by typically a factor from 19.6x to 52.4x with optimal Vdd/Vth and multiple W scaling. In summary, key contributions of the new power minimization technique is: i) without compromising the speed, the total (static and dynamic) power is minimized significantly; ii) with the hierarchical approach, polynomial time optimization is feasible in very large circuits; and iii) the activity-aware delay assignment ensures that the total time slack is maximum and the total power is near-minimal. Future work will include application-specific and architecture-driven issues with this technology scaling techniques.

Table 1. Results of H^{2}TSD-Based Power Optimization

(a) Before Optimization (Fixed Vdd:3.3v, Vth:0.7v)

System Module	Gates/ Depth	Delay (ns)	Input Activity	ϖ, σ_ω	Power Dissipation			
					Leakage	Switching	Short-ckt	Total
4 - Full Adder	106/48	3.36	0.5	17.9, 24.5	2.09x10E-20	4.37x10E-11	2.15x10E-12	4.59x10E-11
			0.05	17.9, 24.5	2.09x10E-20	4.33x10E-12	2.13x10E-13	4.54x10E-12
16 - Look ahead	1838/ 81	7.0	0.5	5.9, 6.2	1.48x10E-19	7.65x10E-10	9.33x10E-11	8.58x10E-10
			0.05	5.9, 6.2	1.48x10E-19	1.39x10E-10	9.29x10E-12	1.48x10E-10
64 - ALU	3417/ 226	18.6	0.5	6.1, 9.2	1.12x10E-18	4.4x10E-09	2.87x10E-10	4.69x10E-09
			0.05	6.1, 9.2	1.12x10E-18	1.90x10E-10	2.87x10E-12	1.93x10E-10
s298	286/18	3.02	0.5	4.8, 7.7	1.92x10E-20	1.44x10E-11	2.37x10E-13	1.46x10E-11
			0.05	4.8, 7.7	1.92x10E-20	1.39x10E-13	2.55x10E-15	1.42x10E-13
s344	229/28	3.86	0.5	15.9, 26.2	4.59x10E-20	6.38x10E-11	9.87x10E-13	6.48x10E-11
			0.05	15.9, 26.2	4.59x10E-20	6.39x10E-13	9.62x10E-15	6.49x10E-13
s386	426/23	3.99	0.5	10.9, 9.2	5.56x10E-20	4.88x10E-11	9.99x10E-13	4.98x10E-11
			0.05	10.9, 9.2	5.56x10E-20	5.13x10E-13	9.62x10E-15	5.23x10E-13
s526	596/18	4.3	0.5	5.2, 7.8	5.88x10E-20	5.13x10E-11	2.00x10E-12	5.33x10E-11
			0.05	5.1, 7.8	5.88x10E-20	5.32x10E-13	9.82x10E-15	5.41x10E-13
c6288	2406/ 129	10.6	0.5	4.7, 8.2	6.52x10E-18	3.21x10E-09	6.55x10E-10	3.87x10E-09
			0.05	4.3, 8.1	6.52x10E-18	4.39x10E-10	6.54x10E-12	4.76x10E-10

(b) After Optimization (Vdd:0.6-1.2v, Vth:0.1-0.52v)

System Module	Hierarchy	Granularity	Input Activity	Vdd,Vth	ϖ, σ_ω	Total Power	Savings	
							Power	Run-Time
4- Full Adder	0	Each gate	0.5	0.6, 0.1	6.61, 8.14	1.95x10E-11	57.5%	0x
			0.05	0.7, 0.1	5.62, 7.19	3.14x10E-13	31.0%	0x
	2	Level 1: 53 Level 2: 17.7	0.5	0.625, 0.1	6.62, 3.14	2.95x10E-11	35.7%	4.28x
			0.05	0.725, 0.2	6.99, 5.64	3.57x10E-13	21.5%	4.28x
	3	Level 1: 53 Level 2: 17.7 Level 3: 2.9	0.5	0.7, 0.12	7.3, 6.22	3.19x10E-11	30.4%	18.8x
			0.05	0.725, 0.2	8.6, 9.0	3.67x10E-13	19.4%	18.8x
16- Look ahead	0	Each gate	0.5	0.8, 0.1	3.1, 2.14	2.93x10E-11	96.6%	0x
			0.05	0.825, 0.1	3.66, 2.94	8.03x10E-13	90.7%	0x
	2	Level 1: 919 Level 2: 306.3	0.5	0.8, 0.1	4.19, 1.14	3.09x10E-11	96.4%	2.26x
			0.05	0.825, 0.1	4.45, 7.10	8.66x10E-13	90.0%	2.26x
	3	Level 1: 919 Level 2: 306.3 Level 3: 51.1	0.5	0.8, 0.1	5.01, 4.14	6.40x10E-11	92.5%	3.08x
			0.05	0.85, 0.12	4.91, 6.16	1.02x10E-12	88.2%	3.08x
64-ALU	0	Each gate	0.5	0.9, 0.1	5.71, 3.13	5.26x10E-11	98.9%	0x
			0.05	0.925, 0.1	5.91, 5.10	2.34x10E-12	98.8%	0x
	2	Level 1: 1708 Level 2: 569.5	0.5	0.925, 0.1	3.63, 3.13	5.50x10E-11	98.8%	2.10x
			0.05	0.95, 0.12	4.62, 5.12	9.60x10E-12	95.0%	2.10x
	3	Level 1: 1708 Level 2: 569.5 Level 3: 94.9	0.5	0.95, 0.12	3.51, 8.15	8.09x10E-11	98.3%	2.70x
			0.05	0.925, 0.2	5.81, 6.14	2.30x10E-11	88.1%	2.70x
s298	0	Each gate	0.5	0.6, 0.1	2.62, 4.44	2.52x10E-13	98.3%	0x
			0.05	0.625, 0.1	3.21, 7.14	4.46x10E-15	96.8%	0x
	2	Level 1: 143 Level 2: 47.7	0.5	0.625, 0.1	3.61, 3.14	5.51x10E-13	96.2%	3.13x
			0.05	0.625, 0.1	3.31, 4.19	1.09x10E-14	92.3%	3.13x
	3	Level 1: 143 Level 2: 47.7 Level 3: 7.94	0.5	0.625, 0.1	4.11, 4.14	8.55x10E-13	94.2%	6.48x
			0.05	0.65, 0.12	4.31, 2.94	1.45x10E-14	89.8%	6.48x
s344	0	Each gate	0.5	0.7, 0.1	8.61, 9.34	6.44x10E-13	99.0%	0x
			0.05	0.725, 0.2	9.21, 3.14	8.31x10E-14	87.2%	0x
	2	Level 1: 115 Level 2: 38.16	0.5	0.8, 0.12	12.1, 5.14	2.03x10E-12	96.9%	3.32x
			0.05	0.8, 0.12	9.61, 2.14	9.36x10E-14	85.6%	3.32x
	3	Level 1: 115 Level 2: 38.16 Level 3: 6.36	0.5	0.85, 0.1	7.61, 3.15	6.02x10E-12	90.7%	7.62x
			0.05	0.825, 0.2	9.61, 2.14	1.35x10E-13	79.2%	7.62x
s386	0	Each gate	0.5	0.6, 0.1	4.61, 5.14	4.43x10E-13	99.1%	0x
			0.05	0.6, 0.1	5.88, 3.74	1.82x10E-14	96.5%	0x
	2	Level 1: 213 Level 2: 71	0.5	0.6, 0.1	7.61, 9.10	4.63x10E-13	99.1%	2.86x
			0.05	0.625, 0.1	7.61, 4.14	1.92x10E-14	96.3%	2.86x
	3	Level 1: 213 Level 2: 71 Level 3: 11.8	0.5	0.625, 0.1	9.33, 4.14	9.58x10E-13	98.1%	5.13x
			0.05	0.65, 0.12	10.01, 9.1	2.16x10E-14	95.9%	5.13x
s526	0	Each gate	0.5	0.6, 0.1	3.61, 3.34	6.91x10E-13	98.7%	0x
			0.05	0.625, 0.1	4.55, 5.15	1.84x10E-14	96.6%	0x
	2	Level 1: 298 Level 2: 99.3	0.5	0.625, 0.1	4.21, 2.14	1.00x10E-12	98.1%	2.68x
			0.05	0.625, 0.1	4.21, 5.94	2.46x10E-14	95.4%	2.68x
	3	Level 1: 298 Level 2: 99.3 Level 3: 16.6	0.5	0.625, 0.1	5.61, 6.14	1.93x10E-12	97.4%	4.39x
			0.05	0.65, 0.12	4.91, 7.14	3.62x10E-14	93.3%	4.39x
c6288	0	Each gate	0.5	0.925, 0.1	5.61, 3.14	3.49x10E-11	99.1%	0x
			0.05	0.9, 0.12	4.67, 4.44	7.10x10E-12	98.5%	0x
	2	Level 1: 1203 Level 2: 401	0.5	0.925, 0.1	3.91, 5.14	5.56x10E-11	98.6%	2.19x
			0.05	0.825, 0.1	5.69, 4.14	7.97x10E-12	98.3%	2.19x
	3	Level 1: 1203 Level 2: 401 Level 3: 66.8	0.5	0.95, 0.2	4.61, 6.99	8.81x10E-11	97.9%	2.90x
			0.05	0.925, 0.2	5.71, 5.54	7.03x10E-11	85.2%	2.90x
Median	0	Each gate	0.5				98.8%	0x
			0.05				96.6%	0x
	2	Level 1: 511 Level 2: 85.1	0.5				97.5%	2.77x
			0.05				93.7%	2.77x
	3	Level 1: 511 Level 2: 85.1 Level 3: 14.2	0.5				95.8%	4.76x
			0.05				88.1%	4.76x
Average	0						90.2%	0x
	2						87.1%	2.86x
	3						83.6%	6.39x

7. REFERENCES

[1] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473-484, April 1992.

[2] J.M. Rabaey and M. Pedram, *Low Power Design Methodologies*, Kluwer Academic Publishers, 1996, pp 21-64,130-160.

[3] R. Nair, C.L. Berman, P.S. hauge, and E.J. Yoffe, "Generation of performance constraints for layout," *IEEE Transactions on Computer-Aided Design*, pp.860-874, Aug. 1989.

[4] T. Gao, P.M. Vaidya, and C.L. Liu,"A new performance driven placement algorithm," *Proc. of ICCAD*, pp. 44-47, 1991.

[5] H. Youssef and E. Shragowitz, " Timing constraints for correct performance," *Proc. of ICCAD*, pp. 24-27, 1990.

[6] P. Pant, V. De, and A. Chatterjee, "Simultaneous power Supply, threshold voltage, and transistor size optimization for low-power operation of CMOS circuits," *IEEE Trans. On VLSI Systems*, vol. 6, no. 4, pp. 538-545, December 1998.

[7] T. Sakurai and A.R. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE Journal Solid-State Circuits*, vol. 25, pp. 584-594, Apr. 1990.

[8] A. Bhavnagarwala, V. De, B. Austin, and J. Meindl, "Circuit techniques for CMOS low power GSI," in Proc. Int. Symp. Low Power Electron. Design: Dig. Tech. Papers, Aug. 1996, pp. 193-196.

[9] A.Raghunathan, N.K. Jha, and S. Dey, *High-Level Power Analysis and Optimization,* Kluwer Academic Publishers, 1998, pp 1-25.

[10] K. Roy and S.C. Prasad, *Low-Power CMOS VLSI Circuit Design,* John wiley & Sons, Inc., 2000, pp. 201-252.

[11] T. Kobayashi and T. Sakurai, "Self adjusting threshold voltage scheme (SATS) for low voltage high speed operation," *IEEE CICC*, 1994, pp.271-277.

[12] S. Mutoh, "1-V Power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 847-, April 1992.

[13] A. Fariborz, "A dynamic threshold voltage MOSFET (DTMOS) for ultra-low voltage operation," IEDM Tech., 1994, pp.809-818.

[14] L. Wei, Z. Chen, and K.Roy, "Double gate dynamic threshold voltage (DGDT) SOI MOSFETs for low power high performance designs," IEEE SOI conference, 1997, pp. 82-83.

[15] S.S. Sapatnekar, V.B. Rao, P.M. Vaidya, and S Kang, "An exact solution to the transistor sizing problem ofr CMOS circuits using convex optimization," *IEEE Trans. On CAD of Integrated Circuits and Systems*, vol. 12, no. 11, pp. 1621-1634, September 1993.